stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 12/73 DECEMBER

J.W. de BAKKER & L.G.L.Th. MEERTENS

ON THE COMPLETENESS OF THE INDUCTIVE ASSERTION METHOD

Prepublication

2e boerhaavestraat 49 amsterdam

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

AMS (MOS) subject classification scheme (1970): 68A05
ACM - Computing Reviews - category: 5.24

On the completeness of the inductive assertion method *)

by

J.W. de Bakker and L.G.L.T. Meertens

ABSTRACT

Manna's theorem on (partial) correctness of programs essentially states that in the statement of the Floyd inductive assertion method: "A flow diagram is correct with respect to given initial and final assertions if suitable intermediate assertions can be found", we may replace "if" by "if and only if". In other words, the method is *complete*.
A precise formulation and proof for the flow chart case is given. The theorem is then extended to programs with (parameterless) recursion; for this the structure of the intermediate assertions has to be refined considerably. The result is used to provide a characterization of recursion which is an alternative to the minimal fixed point characterization, and to clarify the relationship between partial and total correctness.
Important tools are the relational representation of programs, and Scott's induction.

---

*) This paper is not for review; it is meant for publication in a journal.

CONTENTS

1. INTRODUCTION

Our paper describes an investigation in the area of the foundations of program proving. For the statement of the problem we are concerned with, some history is needed.

In [7], Floyd proposed a technique for proving program correctness which later became known as the inductive assertion method. Let us call a program P *correct* with respect to assertions p,q iff for all states x,y, if x satisfies p, and x is mapped by P onto y, then y satisfies q. Floyd's technique can be phrased as: In order to prove the (global) correctness of P with respect to p and q, it is sufficient to find suitable intermediate assertions, and prove the (local) correctness of the program fragments between the intermediate assertions. This method is justified by an inductive argument on the number of times the loops in the program are executed. In several papers by Manna (e.g. [12,13]) Floyd's method was rephrased in the language of (second order) predicate calculus, and the following theorem stated: P is correct (with respect to given p and q) if *and only if* suitable intermediate assertions can be found. This theorem may be viewed as a *completeness* theorem on the inductive assertion method. However, the proofs in [12,13] were not worked out, and, moreover, the theorem was restricted to programs in flow diagram form.

The present paper provides the generalization of the completeness theorem for programs involving recursion, of which, as is well-known, programs in flow diagram form may be considered to be a special case. (The paper by Manna and Pnueli [15] does not give this generalization, since - in the terminology of section 2 - it is concerned with *inclusion* correctness only, the completeness of which is a direct consequence of the minimal fixed point characterization of recursion, see below.)

The construction of the inductive assertions in the case of full recursion is rather more complex than in the flow chart case. In fact, an *infinite* collection of intermediate assertions turns out to be necessary. Structure is brought into this infinity by means of a mechanism which *indexes* the assertions with *traces* reflecting the history of the

computation.

The basic tools to state and prove our completeness theorem (given in section 4) are developed in sections 2 and 3. In section 2 we introduce the relational approach to programming concepts, in particular of sequencing, selection and while statements. The approach allows convenient statement of program correctness, and treatment of the following constructions: Given program P and assertion p, we are interested in: the strongest q such that P is correct with respect to initial p and final q (denoted by $p \circ P$) and: the weakest q such that P is correct with respect to initial q and final p (denoted by $P \rightarrow p$). A number of basic properties of these operations are derived, and they are related to a similar concept introduced by Dijkstra [5]. Section 2 also contains a few remarks on other aspects of the relational approach.

Section 3 introduces (parameterless) recursive procedures. The by now well-known results on their minimal fixed point characterization, leading to Scott's induction rule as important proof rule (as first stated in [19]) are derived again. However, we chose a different approach from e.g. that of [3], by exploiting this time the relationship between a context free grammar and a system of procedure declarations. In particular, we apply the result on context free languages as minimal solutions of systems of equations (e.g. [8]) to the "languages" of elementary actions defined by procedures. Scott's induction rule is illustrated by a short proof of the main theorem of [5].

Section 4 brings the main result of the paper. The completeness theorem for the flow diagram case is first proved by way of introduction; after this, the formalism for its extension to programs with full recursion is developed. An important role is played by the notion of (left- and right-) *companions* of a procedure call, constructs which specify the computation preceding and following an inner call of a procedure within a tree of incarnations of procedures. These companions give the necessary grasp on the history (and future) of the procedure call, and are defined using the indexing mechanism mentioned above. The companions, together with the "$\circ$" and "$\rightarrow$" operations of section 2, are the main tools in the proof of the completeness theorem for which, furthermore, Scott's induc-

tion is essential.

The result is applied in two ways. First of all, an alternative to the minimal fixed point characterization is immediately obtained from it. Secondly, the relationship between the above given notion of correctness (actually called *partial* correctness by Manna) and that of *total* correctness is studied. The completeness theorem is somewhat refined, which then allows the proof of the validity of Manna's reduction of total correctness proofs to proofs in terms of partial correctness.

As remarked at the beginning, the paper is specifically devoted to foundational problems, and not so much to the application of the techniques of section 4 to practical program proving problems (though a few hints on such applications are included at the end of the section).

As related work - besides the already mentioned papers - we should note Engelfriet [6], who is also concerned with completeness results for flow diagrams.

The soundness (not the completeness) of Floyd's method for programs with recursion was already proved in [3].

The present paper is a modification and extension of the technical report [4].

We acknowledge critical comments by M. Fokkinga and W.P. de Roever.

2. PROGRAMS AND RELATIONS

The starting point of the present section is the conception of a program as specification of a *mapping* between *states*. Of course, this view has its limitations, since it abstracts from many properties of the *computation* performed in transforming the states. Therefore, in the next section, in our treatment of recursion, we will have to say more about the connection between the relational and the computational approach.

It is convenient to allow already at the start non-deterministic programs, and to see the mapping P from initial state x to final state y as a binary *relation*, written as $(x,y) \in P$, or, usually, as xPy. Thus, (non-deterministic) programs allow xPy and xPy', with $y \neq y'$.

A slight articulation of the notion of state may be useful. This is done mainly for explanatory reasons, since almost nowhere in the sequel this analysis of the state is really needed.

We view the state - in first approximation - as a mapping from *addresses* - which, called by any other name (ALGOL 68) would work as well - to *values*. As an elementary example, consider the effect of an assignment statement $X_i := f(X_1,X_2,\ldots,X_n)$, where for f one may think of any n-ary function ($n \geq 0$). Suppose that the address (associated with, see remark below) X_i has value a_i, $i=1,2,\ldots,n$. Then we have, in a self-explanatory notation:

$$\begin{pmatrix} X_1 & \ldots & X_i & \ldots & X_n \\ a_1 & \ldots & a_i & \ldots & a_n \end{pmatrix} X_i := f(X_1,\ldots,X_n) \begin{pmatrix} X_1 & \ldots & X_i & \ldots & X_n \\ a_1 & \ldots & f(a_1,\ldots,a_n) & \ldots & a_n \end{pmatrix}$$

Remark: A more refined analysis distinguishes the identifier X_i and the address associated with it, using e.g. *environment* techniques, or the possess-relationship of ALGOL 68. Such refinement is not necessary for our present aim.

Mostly, it will not even be necessary to look as closely on elementary programs as we just did. It suffices to have "elementary actions" A_1, A_2,..., each of which determines - in some way we do not care to

analyze further - a relation between states. The reader may always "fill in" e.g. an assignment statement for such an elementary action, but the structure of that statement will then play no part in our story.

From elementary actions we build up more complex programs with associated relations. Before we go into this, we introduce some notational conventions about operations with relations. Let V be the domain of states, and let R, R_1, R_2,..., be binary relations over V (i.e., subsets of $V\times V$). Then we define

a. Binary operations. *Composition*: $R_1;R_2 = \{(x,y)\,|\,\exists z[xR_1z \wedge zR_2y]\}$. *Union*: $R_1 \cup R_2 = \{(x,y)\,|\,xR_1y \vee xR_2y\}$. *Intersection*: $R_1 \cap R_2 = \{(x,y)\,|\,xR_1y \wedge xR_2y\}$.

b. Unary operation. *Conversion*: $\breve{R} = \{(y,x)\,|\,xRy\}$

c. Nullary operations. The *empty* relation $\Omega = \emptyset$ (the empty subset of $V\times V$). The *identity* relation $I = \{(x,x)\,|\,x \in V\}$. The *universal* relation $U = V\times V$.

d. The star-operation. $R^* = I \cup R \cup R;R \cup \ldots = \bigcup_{i=0}^{\infty} R^i$.

These operations are used in associating relations with programs, or, also, in the formulation of assertions about the correctness of programs.

The programming concepts we treat in this section are: sequencing (denoted by the "go-on" symbol ";"), selection (if ... then ... else) and simple iteration ("while" iteration).

The first concept is immediately taken care of: Let S_1,S_2 be two programs with associated relations R_1,R_2. Then with $S_1;S_2$ we associate the relation $R_1;R_2$.

For selection we need some special measures. Consider the conditional statement if p then S_1 else S_2, where p is some boolean expression (usually called a *predicate* in the sequel). Let the relations p_+ and p_- be defined by: $p_+ = \{(x,x)\,|\,p(x) \text{ is true}\}$, $p_- = \{(x,x)\,|\,p(x) \text{ is false}\}$. It is not difficult to verify that the relation $p_+;R_1 \cup p_-;R_2$ satisfies the usual meaning of the conditional, i.e., $x(p_+;R_1 \cup p_-;R_2)y$ iff $p(x)$ and xR_1y or $\neg p(x)$ and xR_2y.

Observe that for the relations p_+ and p_- we have: $p_+ \cap p_- = \Omega$, $p_+ \cup p_- \subseteq I$, and $p_+ \cup p_- = I$ iff p is a *total* predicate (p is defined for *all* states x). The present notation may take a moment to get used to. As

an exercise, the reader might try to derive e.g. properties of conditionals such as if p then (if p then S_1 else S_2) else S_3 = if p then S_1 else S_3, by proving the equality of the associated relations. (Hint: Use $p_+;p_- = p_+ \cap p_- = \Omega$, and $p;p = p$, for each $p \subseteq I$.)

The next concept we deal with is *iteration*, for the moment only in the form of the while statement while p do S, with the usual semantics: Iterate S as long as p is true (including the case "do nothing" (I!), if p is false to begin with). As corresponding relation we have (assuming, again, that R corresponds to S, this assumption becoming tacit from now on): $(p_+;R)^*;p_-$, also abbreviated as $p * R$.

Remark: Please observe that nothing is alleged to be *proved* here. The treatment is intuitive; a rigorous one follows in the next section - provided the reader is willing to agree that the while loop is a special case of recursion.

The exercises here are: Try to prove, by manipulating with relations: 1) $p * R = p_+;R;p*R \cup p_-$ · 2) $p*(p*R) = p * R$. 3) Let $R * p \stackrel{df}{=} R;p*R$ (representing the *repeat* statement repeat S until $\neg$p). Prove that $R * (p_1 \vee p_2) = (R * p_1) * p_2$.

As the next step one might expect the introduction of the goto statement, either directly, or in the form of a flow diagram specification of the flow of control. Intuitively satisfactory treatment of these is not so easy. Since they are a special case of programs with systems of recursive procedures anyway (more about this in section 4), we do not deal with these separately, but wait till after the introduction of recursion in section 3.

We now continue our relational treatment of programs with the discussion of a number of ways of looking at equivalence and correctness, and their relational representation.

Equivalence is easy: Two programs P_1 and P_2 are *equivalent* iff their associated relations are equal.

The currently most used statement of correctness is the following: A program P is *correct* with respect to the (initial and final) predicates p and q iff

$$\forall x,y\ [p(x) \wedge xPy \rightarrow q(y)] \tag{2.1}$$

i.e., iff for all initial states satisfying p, if P transforms x into y (note that this implies termination of the computation from x to y), then for the final state y, q(y) holds.

This is the formulation which leads to the *inductive assertion method*, as proposed by Floyd and further developed by Manna and Hoare. Relationally, we write for (2.1):

$$p;P \subseteq P;q \tag{2.2}$$

or, more precisely, $p_+;P \subseteq P;q_+$. The + index will be dropped, however, when we expect no confusion to arise; also, instead of p_- we usually will write $\bar{p}$.

We illustrate the form which the inductive assertion method takes by discussion of a simple example; viz. the proof of

$$p;r*P \subseteq r*P;q \tag{2.3}$$

We refer to fig. 1 *).

According to the Floyd technique (which, in essence, was already proposed by Turing in [20]; we owe this reference to R.L. London), we try to find an intermediate assertion s for which we can prove that

$$\begin{cases} p \subseteq s \\ s;r;P \subseteq r;P;s \\ s;\bar{r} \subseteq \bar{r};q \end{cases} \tag{2.4}$$

i.e., in order to prove the *global* fact (2.3), we prove, for suitable s, the *local* facts (2.4), and then infer (2.3).

The soundness of this technique was shown by Floyd by an argument by induction on the number of times the loop is executed. Manna provided the other half by a theorem which - for this special case - amounts to: $p;r*P \subseteq r*P;q$ if *and only if* there exists s such that (2.4) holds. This

*) The figures are collected at the end of the paper.

is Manna's partial correctness theorem [12,13] in its simplest form. In order to explain his treatment of *total* correctness, its formulation has to be refined; we shall return to this at the end of section 4. As remarked in the introduction, the need for a more complete proof of Manna's theorem, together with the desire to generalize it to full recursion, has been the main motivation of the present paper (the other one being the investigation of the relationship between partial and total correctness).

Hoare (almost) writes {p} P{q} for (2.1) [10]. Using this notation, he introduces various *axioms*. E.g., his while statement axiom essentially states again that from (2.4), (2.3) may be inferred. The situation is somewhat different for Hoare's assignment axiom which has the form of something like {p(f(X)} X:=f(X) {p(X)}, i.e. if p(X) is true of the state *after* performing the assignment, then p(f(X)) (the result of substituting f(X) for X in p(X)) was necessarily true *before* its execution. This can be explained by looking again at $(\ldots \overset{X}{a} \ldots)$ X:= f(X) $(\ldots \overset{X}{f(a)} \ldots)$ and noting that p(X) $\leftrightarrow$ p(f(a)) after, and p(f(X)) $\leftrightarrow$ p(f(a)) before the assignment. The reader who is of the opinion that this merits fuller treatment has our sympathy, but that is not the task we have set ourselves in the present paper. We mention this axiom mainly because it has the form of $P;q \subseteq p;P$: if q is true *after* execution of P, then necessarily p had to be true before P. This brings us to a somewhat more systematic treatment of the variants of (2.1), and the way in which the program and one condition together determine (something about) the other condition.

Before we proceed with this, we want to make two remarks.

Firstly, note that both $p;P \subseteq P;q$ and $P;q \subseteq p;P$ are, as many more correctness statements, all special forms of a $P \subseteq Q$ inclusion (e.g., for the first take $xQy \leftrightarrow [p(x) \rightarrow q(y)]$) so that, if one insists, one may view all correctness as simply the inclusion of the relation associated with the program in some other relation.

The second remark is about *termination* (cf. Milner [18]). When we take this in the sense of: P terminates for initial state x iff there exists y such that xPy, we have no problems: We write $\forall x \exists y[xPy]$, or, equivalently, $I \subseteq P;\breve{P}$, and try to prove this for the case at hand. However, sometimes we want to be sure that *all* paths terminate: let P be a

program which terminates, for all input, in this strong sense. Let Q be the nowhere terminating program (L: goto L, say). Let their corresponding relations be R and Ω. Then, though $R \cup \Omega = R$, we object to the conclusion that $P \cup Q = P$ ("$\cup$" taken as programming construct denoting non-deterministic choice), since the left-hand side may, by choosing the second alternative, end up in an unending computation, whereas the right-hand side always terminates. A mechanism for dealing with these problems in terms of the notion of *well-founded* relations, has been proposed and exploited by Hitchcock and Park [9]; we will not pursue these problems further here.

Now back to correctness. We consider once more the formula (2.1)

$$\forall x,y\ [p(x) \wedge xPy \rightarrow q(y)]$$

and observe that it can be written in two other, equivalent, forms: $\forall y\ [\exists x\ [p(x) \wedge xPy] \rightarrow q(y)]$, and $\forall x\ [p(x) \rightarrow \forall y\ [xPy \rightarrow q(y)]]$. This leads us to the introduction of two operations, denoted by "$\circ$" and "$\rightarrow$" respectively:

DEFINITION 2.1

$$(p \circ P)(x) \leftrightarrow \exists y[p(y) \wedge yPx]$$
$$(P \rightarrow p)(x) \leftrightarrow \forall y[xPy \rightarrow p(y)]$$

Remark: This definition includes the "extreme" cases $p = \Omega$ and $p = I$, standing for the identically false and true predicate, respectively. From these definitions we immediately infer the following lemma:

LEMMA 2.1

1. $p;P \subseteq P;(p \circ P)$
   $(P \rightarrow q);P \subseteq P;q$
2. For all p,q, if $p;P \subseteq P;q$, then $p \circ P \subseteq q$, and $p \subseteq P \rightarrow q$.
3. $p \circ P = \cap \{q \mid p;P \subseteq P;q\}$
   $P \rightarrow q = \cup \{p \mid p;P \subseteq P;q\}$

PROOF. Parts 1 and 2 follow from the definitions, part 3 from parts 1 and 2. □

We will also have occasion to use the operations $p \circ \breve{P}$ and $\breve{P} \to q$, for which we have $p \circ \breve{P} = \cap \{q \mid p;\breve{P} \subseteq \breve{P};q\} = \cap \{q \mid P;p \subseteq q;P\}$, and $\breve{P} \to q = \cup \{p \mid P;p \subseteq q;P\}$. (Observe that we used here that $P \subseteq Q \leftrightarrow \breve{P} \subseteq \breve{Q}$, $\widetilde{P_1;P_2} = \breve{P}_2;\breve{P}_1$, and $\breve{p} = p$ for $p \subseteq I$.)

The basic properties of the "$\circ$" and "$\to$" operations are collected in lemma's 2.2 and 2.3.

LEMMA 2.2

1. $\Omega \circ P = P \circ \Omega = \Omega$
2. $P;I \circ P = P$
3. $p \circ q = p;q = p \cap q$
4. $p \circ (P_1;P_2) = (p \circ P_1) \circ P_2$
5. $p \circ (P_1 \cup P_2) = (p \circ P_1) \cup (p \circ P_2)$
6. If $P_1 \subseteq P_2$ then $p \circ P_1 \subseteq p \circ P_2$
7. If $p \subseteq q$, then $p \circ P \subseteq q \circ P$
8. $(p \cup q) \circ P = (p \circ P) \cup (q \circ P)$
9. If $\breve{P}$ is a function, then $(p \cap q) \circ P = (p \circ P) \cap (q \circ P)$

PROOF. The proofs are immediate from the definitions. We prove only parts 2 and 4:

2. $\forall x,y\,[xP;\ I \circ Py \leftrightarrow xPy \wedge (I \circ P)(y) \leftrightarrow xPy \wedge \exists z[I(z) \wedge zPy] \leftrightarrow$
$\leftrightarrow xPy \wedge \exists z[zPy] \leftrightarrow xPy]$

4. $\forall x\,[(p \circ (P_1;P_2))(x) \leftrightarrow \exists y[p(y) \wedge y\ P_1;P_2\ x] \leftrightarrow$
$\leftrightarrow \exists y,z[p(y) \wedge yP_1z \wedge zP_2x] \leftrightarrow$
$\leftrightarrow \exists z[\exists y[p(y) \wedge y\ P_1z] \wedge zP_2x] \leftrightarrow$
$\leftrightarrow \exists z[(p \circ P_1)(z) \wedge zP_2x] \leftrightarrow ((p \circ P_1) \circ P_2)(x)]$. □

For "$\to$" we have similar properties, some of which are mentioned in

LEMMA 2.3

1. $P \to I = I,\ I \to p = p$
2. $I \subseteq (p_1 \to p_2)$ iff $p_1 \subseteq p_2$
3. $(P \to \Omega);P = \Omega$
4. $(P_1;P_2) \to p = (P_1 \to (P_2 \to p))$
5. $(P_1 \cup P_2) \to p = (P_1 \to p) \cap (P_2 \to p)$

PROOF. Immediate. □

When we compare $p \circ \breve{P} = \cap \{q \mid P;p \subseteq q;P\}$, and $P \rightarrow p = \cup \{q \mid q;P \subseteq P;p\}$, the question arises as to when these constructs coincide. The answer is given in terms of the notions of *functionality* and *totality* of P: P is a function iff $\breve{P};P \subseteq I$, or, equivalently, $\forall x,y,z[xPy \wedge xPz \rightarrow y=z]$. P is total iff $I \subseteq P;\breve{P}$ or, equivalently, $\forall x \exists y[xPy]$. We then have:

LEMMA 2.4

1. If P is a function, then $p \circ \breve{P} \subseteq P \rightarrow p$.
2. If, for all p, $p \circ \breve{P} \subseteq P \rightarrow p$, then P is a function.
3. If P is total, then $P \rightarrow p \subseteq p \circ \breve{P}$.
4. If, for all p, $P \rightarrow p \subseteq p \circ \breve{P}$, then P is total.
5. (Conclusion) P is a total function iff $\forall p\ [P \rightarrow p = p \circ \breve{P}]$.

PROOF. We show only part 2. Its assumption is equivalent to: $\forall p\ [\forall x,y[xPy \wedge p(y) \rightarrow \forall z[xPz \rightarrow p(z)]]]$. Let y_0 be some element in the range of P, and let $p(y) \leftrightarrow y = y_0$. Then we see that the assumption amounts to: If xPy and $y = y_0$ and xPz, then $z = y_0$; hence, P is indeed a function. □

It is perhaps of some interest to compare our operations with a notion used by Dijkstra [5], from which we quote: "We consider the semantics of a program P fully determined when we can derive for any post-condition p to be satisfied by the final state, the weakest precondition that for this purpose should be satisfied by the initial state. We regard this weakest precondition as a function of the post-condition p and denote it by $f_P(p)$."

This suggests to us that what is meant here is that $f_P(p) = P \rightarrow p = \cup \{q \mid q;P \subseteq P;p\}$. The use of $f_P(p)$ in the cited paper furthermore seems to imply that satisfaction of $f_P(p)$ guarantees termination, i.e., that $f_P(p)$ should be taken as $f_P(p) = (I \circ \breve{P}) \cap (P \rightarrow p)$, or, equivalently, $f_P(p) = (p \circ \breve{P}) \cap (P \rightarrow p)$. Dijkstra also imposes the restriction that P is a function, in which case $f_P(p)$ reduces to $p \circ \breve{P}$ (cf. lemma 2.4). With these restrictions, the axioms postulated in [5] become provable, and are in fact parts 1, 4, 7, 8 and 9 of lemma 2.2. If we omit the requirement that we deal with functions only, properties 1, 7 and 9 remain valid, but properties 4 and 8 have to be modified as follows:

4'. $f_{P_1;P_2}(p) \subseteq f_{P_1}(f_{P_2}(p))$
(with equality only if P_1 is a function)

8'. $f_P(p \cup q) \supseteq f_P(p) \cup f_P(q)$
(with equality only if P is a function)

A further remark on $f_P(p)$ will follow at the end of section 3.

Since we are working in a relational framework, a relational version of the "$\circ$" and "$\rightarrow$" operations may be of interest. For "$\circ$" this can be given directly, but for "$\rightarrow$" we have to use complementation of relations with respect to I: For $p \subseteq I$, $\bar{p} \stackrel{df}{=} I \setminus p$.

LEMMA 2.5

1. $p \circ P = U;p;P \cap I$
(Remember that U is the universal relation.)
2. $P \rightarrow p = \overline{\breve{P} \circ \bar{p}}$

PROOF. Left to the reader. □

As final lemma we need

LEMMA 2.6

$R_1 \subseteq R_2$ iff $\forall p,q[$if $p;R_2 \subseteq R_2;q$, then $p;R_1 \subseteq R_1;q]$.

PROOF. $\Rightarrow$is obvious. As to $\Leftarrow$: Choose some fixed x_0, and assume $x_0 R_1 y$. Choose, furthermore, $p_0(x) \leftrightarrow x = x_0$ and $q_0(y) \leftrightarrow x_0 R_2 y$. Then $p_0;R_2 \subseteq R_2;q_0$ holds; hence, $p_0;R_1 \subseteq R_1;q_0$ follows, i.e., $x = x_0 \wedge x\ R_1 y \rightarrow x_0 R_2 y$. Thus, the assumption $x_0 R_1 y$ leads to $x_0 R_2 y$. Since x_0 was arbitrary, the proof is completed. □

COROLLARY 2.6. If, for all p and q, R_1 is correct with respect to p and q iff R_2 is correct with respect to p and q, then $R_1 = R_2$. (Compare this with: If, for all Q, R_1 is correct with respect to Q ($R_1 \subseteq Q$) iff R_2 is correct with respect to Q, then $R_1 = R_2$.)

PROOF. Direct from lemma 2.6. □

As an exercise to conclude this section we offer to the reader who is insufficiently challenged by our elementary lemma's: Let $R^{\dagger} \stackrel{df}{=} (I \circ \breve{R}) * R$, i.e., perform R as long as it is defined (e.g., if R is the descendent relation in a tree, $R^{\dagger}$ connects the root with all leaves.). Prove that $R^{\dagger\dagger\dagger\dagger} = R^{\dagger\dagger}$.

3. RELATIONS AND RECURSION

The relational approach to program semantics is now extended to programs involving recursion.

Our treatment of this is not essentially different from e.g. that of [3], and may be skipped by the reader who knows already about procedures as minimal fixed points and Scott's induction, and who wants to proceed immediately with the main results of our paper in the next section. However, a number of points are stressed differently, e.g., the systematic distinction between language and interpretation is kept in the background here. Moreover, the main result - procedures as minimal fixed points with corresponding induction rule - is now obtained by exploiting the correspondence between systems of recursive procedures and context free grammars (cf. also [1]). This has the advantage - besides the obvious one of clarification of the correspondence - that we can rely on a well-known result in formal language theory, stating that context free languages are minimal solutions of systems of equations, and, moreover, that these solutions are obtained by successive approximations (see e.g. Ginsburg [8]; this result may be seen as an instance of Kleene's first recursion theorem [11]).

In a program with recursion we have a system of (mutually recursive) procedure declarations, together with what may be called the "main" statement of the program, which, normally, contains calls of the declared procedures. Both this statement and the statements of the procedure bodies are supposed to have the structure as introduced in the previous section. That is, they are made up from elementary actions, to which now the procedure symbols are added, by means of composition and union (the last construct modelling conditionals).

More formally, a (recursive) program T consists of a set of declarations $\mathcal{D} = \{P_1 \Leftarrow S_1, P_2 \Leftarrow S_2, \ldots, P_n \Leftarrow S_n\}$, and a statement S; i.e., $T = (\mathcal{D}, S)$. Here "$\Leftarrow$" stands for "is recursively defined by" (in ALGOL 60 we would write procedure $P_i; S_i$, $i=1,2,\ldots,n$). Observe that $\mathcal{D}$ is a *set* since the order in which the declarations are given will turn out to be immaterial.

Often, we want to emphasize that the S_i, $i=1,\ldots,n$, or S, may contain occurrences of the P_i, $i=1,\ldots,n$, and we write $S_i = S_i(P_1,P_2,\ldots,P_n)$,

$S = S(P_1,P_2,\ldots,P_n)$. This notation is also used in the customary way for indicating substitution: The result of simultaneously substituting, in S, for each P_j the statement $S^{(j)}$, j=1,2,...,n, is denoted by $S(S^{(1)},S^{(2)},\ldots,S^{(n)})$.

Before we proceed with a more detailed formulation of the structure of the S_i, one comment may be in order. The reader will have noted that our procedures are *parameterless*. Admittedly, this is a restriction which leaves out of consideration some interesting (and difficult) problems. However, we are of the opinion that a satisfactory treatment of the various ways of parameter passing cannot be given without the introduction of (the equivalent of) the ALGOL 68 notions of identity declaration and proceduring, an idea which is not pursued in the present paper. In defense of the restriction, I can only remark that first of all there *is* a correspondence (given below) between parameterless procedures and the *monadic* recursive function schemes of e.g. [1], and, secondly, that it will appear - hopefully - that even parameterless procedures lead to some interesting considerations which, moreover, are needed anyhow in order to fully understand procedures *with* parameters. So far for the apology.

Now we continue with a precise definition of the class of recursive programs.

We start with the class $A = \{A_1,A_2,\ldots\}$ of elementary actions, $B = \{p,p_1,\bar{p},\ldots,q,\ldots,r,\ldots\}$ of booleans, and $C = \{I,\Omega\}$ of constants. (Remember that I denotes the identity (dummy) statement and Ω the empty statement.) Let $R = A \cup B \cup C$, and P (the class of procedure symbols) = $= \{P_1,P_2,\ldots\}$. Then the class of statements over R and P, denoted by $S(R,P)$, is defined by

1. $R \cup P \subseteq S(R,P)$
2. If $S_1,S_2 \in S(R,P)$ then $(S_1;S_2)$ and $(S_1 \cup S_2) \in S(R,P)$.

Examples of programs are:

1. $(\{P \Leftarrow ((p;(A;P)) \cup \bar{p})\}, P)$
2. $(\{P_1 \Leftarrow ((p;P_2) \cup \bar{p}), P_2 \Leftarrow ((p;(A;P_2)) \cup (\bar{p};P_1))\}, P_1)$.

Anticipating the analysis given below, the reader may already observe that for the P of the first example we have that $P = p * A$, and for the

P_1 of the second example: $P_1 = p * (p*A)$. Moreover, as corresponding monadic function schemes we have:

1. $f(x) \Leftarrow$ **if** $p(x)$ **then** $f(a(x))$ **else** x
2. $f_1(x) \Leftarrow$ **if** $p(x)$ **then** $f_2(x)$ **else** x
   $f_2(x) \Leftarrow$ **if** $p(x)$ **then** $f_2(a(x))$ **else** $f_1(x)$

Clearly, our definition of the class of programs causes some parentheses trouble. However, our formal treatment does need their introduction, in order that we can later *prove* that we may drop them unambiguously.

Our task is now to find the relations corresponding to procedures, just as we did this before for the constructs of sequencing, selection and simple iteration. As before, we assume known how the elementary actions are executed, and now have to analyze how a program, for given initial state $x = x_0$, determines a sequence of elementary actions applied successively to intermediate states x_i, eventually leading to the final state $x_n = y$. In this analysis, the notion of *computation point* plays a useful role:

DEFINITION 3.1. A *computation point* is a triple (S_ℓ, x, S_r), where S_ℓ is a sequence of zero or more elementary actions (the empty sequence being identified with the identity I), x is some state, and S_r is some statement in $S(R,P)$.

Intuitively, a computation point (S_ℓ, x, S_r) denotes, at each moment of the computation, that

1. S_ℓ is the sequence of elementary actions already performed.
2. x is the current state.
3. S_r is the remainder of the program which still awaits execution.

Using this notion, the definition of a computation prescribed by a program $T = (\mathcal{D}, S)$, when applied to initial state x, follows rather naturally: Begin with initial computation point (I, x, S), ($S_\ell = I$: nothing has yet been executed), define the allowed transitions between the computation points in accordance with the intended meaning of the various program constructs, and then end up with some final (S', y, I), with S' some sequence of elementary actions, y the final state, and $S_r = I$ indicating that nothing remains to be done.

So we need to define the allowed transitions between computation points:

DEFINITION 3.2. Let $\mathcal{D}$ be a set of declarations. A computation step is a *$\mathcal{D}$-allowed transition* between two computation points (S_ℓ,x,S_r) and (S'_ℓ,x',S'_r) iff one of the conditions 1a,1b,2a,...,2e, is satisfied.

1a. $S_r = (R;S'_r)$, for some $R \in \mathcal{R}$, and, moreover, xRx', and $S'_\ell = S_\ell;R$ hold.
(Observe that this implies that $R \neq \Omega$, and that, if $R = p \in \mathcal{B}$, then xpx', or, equivalently, $x = x'$ and $p(x)$ hold.)

1b. $S_r = R$, for some $R \in \mathcal{R}$, $S'_r = I$, and, moreover, xRx' and $S'_\ell = S_\ell;R$ hold.

2a. $S_r = ((S_{r,1};S_{r,2});S_{r,3})$, $S'_r = (S_{r,1};(S_{r,2};S_{r,3}))$,
$S'_\ell = S_\ell$, $x' = x$.

2b. $S_r = (S_{r,1} \cup S_{r,2})$, $S'_r = S_{r,1}$ or $S'_r = S_{r,2}$,
$S'_\ell = S_\ell$, $x' = x$.

2c. $S'_r = ((S_{r,1} \cup S_{r,2});S_{r,3})$, $S'_r = ((S_{r,1};S_{r,3}) \cup (S_{r,2};S_{r,3}))$,
$S'_\ell = S_\ell$, $x' = x$.

2d. $S_r = (P;S_{r,1})$, $S'_r = S;S_{r,1}$, where $P \Leftarrow S \in \mathcal{D}$,
$S'_\ell = S_\ell$, $x' = x$.
(Observe that the replacement of the procedure identifier P by its body S is the usual copy rule of procedure semantics.)

2e. $S_r = P$, $S'_r = S$, where $P \Leftarrow S \in \mathcal{D}$,
$S'_\ell = S_\ell$, $x' = x$.

Example (not bothering for the moment about parentheses):
A sequence of $\mathcal{D}$-allowed computation steps, where $\mathcal{D}$ is $\{P \Leftarrow p;A;P \cup \bar{p}\}$, is
(I,x_0,P), $(I,x_0,p;A;P \cup \bar{p})$, $(I,x_0,p;A;P)$, $(p,x_1,A;P)$, $(p;A,x_2,P)$, $(p;A,x_2,p;A;P \cup \bar{p})$, $(p;A,x_2,p;A;P)$, $(p;A;p,x_3,A;P)$, $(p;A;p;A,x_4,P)$, $(p;A;p;A,x_4,p;A;P \cup \bar{p})$, $(p;A;p;A,x_4,\bar{p})$, $(p;A;p;A;\bar{p},x_5,I)$
where

1. $p(x_0)$ and $p(x_2)$ are true, $p(x_4)$ is false;
2. $x_0 = x_1$, $x_1 A x_2$, $x_2 = x_3$, $x_3 A x_4$, and $x_4 = x_5$ hold;
3. we have omitted - as we will do in the sequel - the identity action in a sequence of elementary actions.

The definition of the relation to be associated with program $T = (\mathcal{D},S)$

should give no problem:

DEFINITION 3.3. Let $T = (\mathcal{D},S)$ be a program. Then $(x,y) \in T$ iff there exists a sequence of elementary actions S', and a sequence of $\mathcal{D}$-allowed computation steps from (I,x,S) to (S',y,I).

From now on, we assume the set $\mathcal{D}$ of declarations fixed, unless otherwise stated, and we write xSy instead of $x(\mathcal{D},S)y$. Also, we understand $S_1 \subseteq S_2$ or $S_1 = S_2$ with reference to this $\mathcal{D}$.

From definition 3.3, a number of properties follow rather directly, reason why we omit their proofs.

LEMMA 3.1

1. $((S_1;S_2);S_3) = (S_1;(S_2;S_3))$ $(= S_1;S_2;S_3$, from now on)
2. $S_1 \cup S_2 = S_2 \cup S_1$
3. $((S_1 \cup S_2);S_3) = ((S_1;S_3) \cup (S_2;S_3))$ (this will, by convention, be written as $S_1;S_3 \cup S_2;S_3$)
4. Similar for left-distributivity of ";" over "$\cup$".
5. If $P \Leftarrow S \in \mathcal{D}$, then $P = S$. (This is the *fixed point property* of procedures.)
6. $x\ S_1;S_2\ y$ (according to definition 3.3) iff $\exists z[xS_1z \wedge zS_2y]$
   $x\ S_1 \cup S_2\ y$ (according to definition 3.3) iff $xS_1y \vee xS_2y$
   (i.e., definition 3.3 is a consistent extension of the definitions of ";" and "$\cup$" of section 2).
7. (Monotonicity) If $S_i^{(1)} \subseteq S_i^{(2)}$, $i=1,2,\ldots,n$, then $S(S_1^{(1)},\ldots,S_n^{(1)}) \subseteq S(S_1^{(2)},\ldots,S_n^{(2)})$.
8. If (S_ℓ,x,S_r), (S'_ℓ,x',S'_r) is a $\mathcal{D}$-allowed computation step, then $S_\ell;S_r \supseteq S'_\ell;S'_r$.
9. $S = \bigcup\{S' \mid \exists x,y$ such that $(I,x,S),\ldots,(S',y,I)$ is a sequence of $\mathcal{D}$-allowed computation steps$\}$

These facts being - as we hope - satisfactorily established by the reader, we now continue with the refinement of the analysis, leading up to the *minimality* of the fixed points.

We start with the following two observations:

1. The four-tuple $(P,R,\mathcal{D},S)$ reminds one of a context free grammar, with P: non terminals, R: terminals, $\mathcal{D}$: productions rules, and S: start

symbol.

2. The way in which the $\mathcal{D}$-allowed computation steps are defined - in particular the procedure-call step 2d,e - reminds one of the production steps in the derivation of a context free language.

To this we add the following by way of further introduction: Consider the procedure P defined (on the natural numbers) by: P $\Leftarrow$ if x=0 then x:=0 else (x:=x-1;P;x:=x+1). Our assertion that $P = \bigcup_{n=0}^{\infty} ((x:=x-1)^n; x:=0; (x:=x+1)^n)$ will - after some thought - not be surprising, nor the similarity of this expression with the "language" $\{(x:=x-1)^n;x:=0;(x:=x+1)^n \mid n \geq 0\}$. We now make these informal observations more precise.

Let τ be a mapping from statements S in $S(R,P)$ to subsets of the language $(A \cup B \cup P)^*$ - i.e., the set of all finite (possible empty) sequences of symbols in A,B or P -, defined as follows (identifying singleton sets with their elements):

$\tau(A) = A$, $\tau(p) = p$, $\tau(P) = P$

$\tau(S_1;S_2) = \tau(S_1)\tau(S_2)$ (juxtaposition denoting the usual "product" of sets of words).

$\tau(S_1 \cup S_2) = \tau(S_1) \cup \tau(S_2)$,

$\tau(\Omega) = \emptyset$ (the empty set of words),

$\tau(I) = \varepsilon$ (the empty word).

For $\mathcal{D} = \{P_i \Leftarrow S_i\}_{i=1}^n$, we define $\tau(\mathcal{D}) = \{P_i \to S_i' \mid i=1,2,\ldots,n$ and $S_i' \in \tau(S_i)\}$. Then for the program $(\mathcal{D},S)$ we have as corresponding grammar $(\tau(P), \tau(A \cup B), \tau(\mathcal{D}), \tau(S))$. Note that there is a slight generalization involved, in that $\tau(S)$ is, in general, not just an element of $\tau(P)$ (the set of non-terminals), but a subset of $(\tau(P \cup A \cup B))^*$.

Example: For the program $(\{P \Leftarrow p;A;P \cup \bar{p}\}, P \cup A)$ we have as corresponding grammar: $(\{P\}, \{p,\bar{p},A\}, \{P \to pAP, P \to \bar{p}\}, \{P,A\})$.

The next definition introduces the language associated with a program.

DEFINITION 3.4. Let $T = (\mathcal{D},S)$ be a program. Let $\tau(T) = (\tau(P), \tau(A \cup B), \tau(\mathcal{D}), \tau(S))$ be the (generalized) context free grammar associated with T. Then

$$L(\tau(T)) = \{S'' \mid S'' \in (\tau(A \cup B))^* \text{ and } \exists S' \in \tau(S) \text{ such that } S' \overset{*}{\underset{\tau(T)}{\Rightarrow}} S''\}$$

where $\overset{*}{\underset{\tau(T)}{\Rightarrow}}$ is defined in the usual way as derivation with respect to the grammar $\tau(T)$.

Example: For $T = (\{P \Leftarrow p;A;P \cup \bar{p}\}, P)$, we have $L(\tau(T)) = \{(pA)^i\bar{p} \mid i \geq 0\}$.

So far everything was rather straightforward. The next step also seems clear: One might at first expect that the set of all elementary actions determined by a program on the base of definition 3.3, coincides with the language of definition 3.4. There is a slight complication, however. Example: $T = (\{P \Leftarrow p;A_1 \cup \bar{p};A_2\}, p;P)$. Then $L(\tau(T)) = (ppA_1, p\bar{p}A_2)$, but there is no x,y such that $(I,x,p;P),\ldots,(p;\bar{p};A_2,y,I)$ is an allowed sequence of computation steps.

This is easily taken care of, however, by noting that those sequences of $L(\tau(T))$ which do not occur as possible computations are necessarily equivalent with Ω. Using the notation $\tau^{-1}(L)$ for $\bigcup_{\tau(S)\in L} S$ (this yields one relation, not a set of relations!) we have as

LEMMA 3.2. $T = \tau^{-1}(L(\tau(T)))$.

PROOF. Direct from the definitions. □

Continuing with the last example, $L(\tau(T)) = \{ppA_1, p\bar{p}A_2\}$. Hence, $\tau^{-1}(L(\tau(T))) = p;p;A_1 \cup p;\bar{p};A_2 = p;p;A_1 \cup \Omega = p;p;A_1 = T$.

We have now reached the point where we can apply the result of e.g. [8], stating that context free languages are minimal solutions of a system of equations, which solutions are obtainable by means of successive approximations, starting from the empty set.

Let $\mathcal{D} = \{P_i \Leftarrow S_i\}_{i=1}^n$, let $S = S(P_1,\ldots,P_n)$ be an element of $S(R,P)$, and let $T = (\mathcal{D},S)$. By the definition of τ, $\tau(S) = \tau(S)(P_1,\ldots,P_n)$ (i.e., $\tau(S)$ is a set of words, each of which may contain occurrences of $P_1,\ldots,P_n$).

Let $\tau(S)^{[0]} \overset{df}{=} \emptyset$

$\tau(S)^{[j+1]} \overset{df}{=} \tau(S)(\tau(S_1)^{[j]},\ldots,\tau(S_n)^{[j]})$, $j=0,1,\ldots$.

Then, by [8], $L(\tau(T)) = \bigcup_{j=0}^{\infty} \tau(S)^{[j]}$. Hence by lemma 3.2,

$$T = \tau^{-1}(L(\tau(T)) = \tau^{-1}(\bigcup_{j=0}^{\infty} \tau(S)^{[j]}).$$

Now let $S^{(0)} \stackrel{df}{=} \Omega$, $S^{(j+1)} \stackrel{df}{=} S(S_1^{(j)},\ldots,S_n^{(j)})$.

Then it is not difficult to verify that $S^{(j)} = \tau^{-1}(\tau(S)^{[j]})$, and, moreover,

$$\bigcup_{j=0}^{\infty} S^{(j)} = \bigcup_{j=0}^{\infty} \tau^{-1}(\tau(S)^{[j]}) = \tau^{-1}(\bigcup_{j=0}^{\infty} \tau(S)^{[j]}) = T.$$

Thus, we have $T = (\mathcal{D},S) = \tau^{-1}(\bigcup_{j=0}^{\infty} \tau(S)^{[j]}) = \bigcup_{j=0}^{\infty} S^{(j)}$.

Once more omitting reference to $\mathcal{D}$ this yields

THEOREM 3.1 (The union theorem for programs with recursive procedures).

$$S = \bigcup_{j=0}^{\infty} S^{(j)}.$$

COROLLARY 3.1. Let $\mathcal{D} = \{P_i \Leftarrow S_i\}_{i=1}^{n}$, and let Q_i satisfy $S_i(Q_1,\ldots,Q_n) \subseteq Q_i$, $i=1,2,\ldots,n$. Then $P_i \subseteq Q_i$.

PROOF. We use that $P_i = S_i$ (lemma 3.1.5), and that $S_i = \bigcup_{j=0}^{\infty} S_i^{(j)}$. Then using induction on j, for each $i=1,2,\ldots,n$,

$$S_i^{(0)} = \Omega \subseteq Q_i$$
$$S_i^{(j+1)} = S_i(S_1^{(j)},\ldots,S_n^{(j)}) \subseteq S_i(Q_1,\ldots,Q_n) \subseteq Q_i$$
(by monotonicity (lemma 3.1.7) and the induction hypothesis).

Thus, $\bigcup_{j=0}^{\infty} S_i^{(j)} \subseteq Q_i$, whence $P_i \subseteq Q_i$, $i=1,\ldots,n$, follows. □

COROLLARY 3.2. (Minimal fixed point property). Let $\mathcal{D}$ be as before. Then
$(P_1,\ldots,P_n) = \bigcap\{(Q_1,\ldots,Q_n) \mid S_i(Q_1,\ldots,Q_n) = Q_i,\ i=1,\ldots,n\}$.

PROOF. By lemma 3.1.5 and corollary 3.1, the P_i are fixed points of the S_i which are included in all fixed points; hence, they are minimal fixed points. □

The next corollary is an easy consequence of corollary 3.2, and deals with correctness in terms of inclusion ($P \subseteq Q$); it is stated for compari-

son with similar results to be given in section 4, for correctness in terms of assertions $(p;P \subseteq P;q)$:

COROLLARY 3.3. Let $\mathcal{D} = \{P_i \Leftarrow S_i\}_{i=1}^n$

1. (Correctness in terms of inclusion)

$$\forall j=1,\ldots,n, Q_j[P_j \subseteq Q_j \text{ iff } \exists Q_1',\ldots,Q_n'[S_i(Q_1',\ldots,Q_n') \subseteq Q_i', i=1,\ldots,n, \text{ and } Q_j' \subseteq Q_j]].$$

2. (Characterizing recursion in terms of inclusion correctness)

$$\forall R_1,\ldots,R_n$$
$$[\forall j=1,\ldots,n, Q_j[R_j \subseteq Q_j \text{ iff } \exists Q_1',\ldots,Q_n'[S_i(Q_1',\ldots,Q_n') \subseteq Q_i', i=1,\ldots,n, \text{ and } Q_j' \subseteq Q_j]]$$
$$\text{iff}$$
$$\forall j=1,\ldots,n\ [R_j = P_j]].$$

PROOF. Part 1 follows from corollary 3.2, and part 2 from part 1. □

The next main application of the union theorem is in the proof of Scott's induction rule, which plays an important part in section 4 (and elsewhere in proofs about recursion, see e.g. [2,3,14,16]).

THEOREM 3.2 (Scott's induction rule).
Let $\mathcal{D} = \{P_i \Leftarrow S_i\}_{i=1}^n$. Let $S_\ell = S_\ell(P_1,\ldots,P_n)$ and $S_r = S_r(P_1,\ldots,P_n)$ be two statements satisfying the two conditions:

1. $S_\ell(\Omega,\ldots,\Omega) \subseteq S_r(\Omega,\ldots,\Omega)$.
2. If $S_\ell(X_1,\ldots,X_n) \subseteq S_r(X_1,\ldots,X_n)$, then
$S_\ell(S_1(X_1,\ldots,X_n),\ldots,S_n(X_1,\ldots,X_n)) \subseteq$
$S_r(S_1(X_1,\ldots,X_n),\ldots,S_n(X_1,\ldots,X_n))$;

Then we have that $S_\ell = S_\ell(P_1,\ldots,P_n) \subseteq S_r(P_1,\ldots,P_n) = S_r$.

PROOF. Let, as before, for $S \in S(R,P)$, $S^{(0)} = \Omega$, $S^{(j+1)} = S(S_1^{(j)},\ldots,S_n^{(j)})$. By condition 1, we see that $S_\ell^{(1)} \subseteq S_r^{(1)}$. Then, using condition 2 (with Ω for X_i), we infer that $S_\ell(S_1(\Omega,\ldots,\Omega),\ldots,S_n(\Omega,\ldots,\Omega)) \subseteq S_r(S_1(\Omega,\ldots,\Omega),\ldots,S_n(\Omega,\ldots,\Omega))$, i.e., that $S_\ell^{(2)} \subseteq S_r^{(2)}$. Repeating this argument we obtain that $S_\ell^{(j)} \subseteq S_r^{(j)}$, $j=0,1,2,\ldots$, and the desired conclusion $S_\ell = \bigcup_{j=0}^{\infty} S_\ell^{(j)} \subseteq \bigcup_{j=0}^{\infty} S_r^{(j)} = S_r$ follows by the union theorem. □

Remark: The induction theorem is easily seen to go through for sets of inclusions instead of for just one inclusion $S_\ell \subseteq S_r$.

Example of applying the rule: In [5], Dijkstra notes the following fact: Let $\mathcal{D} = \{P \Leftarrow S(P)\}$, and let q,r be arbitrary predicates. Let, as before, $f_P(q)$ mean $q \circ \breve{P}$ (cf. section 2; we assume that the restriction that the P are functions is again imposed). Suppose the following condition is satisfied: If $q \subseteq f_X(r)$ then $q \subseteq f_{S(X)}(r)$. Then we may conclude (according to Dijkstra), that $q \cap f_P(I) \subseteq f_P(r)$. He calls this the "Fundamental Invariance Theorem for Recursive Procedures", and spends some effort in proving this. In the form as given, it is wrong, however. This is easily seen by taking $r = \Omega$: The condition "If $q \subseteq f_X(\Omega)$ then $q \subseteq f_{S(X)}(\Omega)$", is then trivially satisfied, since it reduces to (lemma 2.2.1) "if $q \subseteq \Omega$ then $q \subseteq \Omega$". This would therefore imply that for arbitrary q and P, $q \cap f_P(I) \subseteq \Omega$, which is obviously absurd. (Taking, e.g., $q = I$ would yield that no procedure ever terminates.)
With the following amended version of the condition: If $q \cap f_X(I) \subset f_X(r)$ then $q \cap f_{S(X)}(I) \subseteq f_{S(X)}(r)$, the theorem is an easy consequence of the induction theorem:

1. $q \cap f_\Omega(I) \subseteq f_\Omega(r)$ is clear from lemma 2.2.
2. "If $q \cap f_X(I) \subseteq f_X(r)$ then $q \cap f_{S(X)}(I) \subseteq f_{S(X)}(r)$" is precisely the second condition of the induction theorem which we need to conclude that $q \cap f_P(I) \subseteq f_P(r)$.

Thus, we see that the (corrected) example is easily dealt with by means of the induction theorem.

4. RECURSION AND INDUCTIVE ASSERTIONS

This section brings the generalization of Manna's treatment of partial (and total) correctness, and an application of the result providing an alternative characterization of recursion, using a certain property expressed in terms of inductive assertions instead of the minimal fixed point property used in corollary 3.3.

The main tool of the section consists in the enrichment of the inductive assertion method with an *indexing* of the assertions in such a way that the index can be considered as a *trace* of the history of the computation. Such rather complex structuring of the assertions turns out to be necessary for the only if part of the theorem: $p;P \subseteq P;q$ if and *only if* suitable intermediate assertions can be found.

In order to bring out the difficulty, we consider once more formulae (2.3) and (2.4). We saw that: if $\exists s[p \subseteq s,\ s;r;S \subseteq r;S;s,\ s;\bar{r} \subseteq \bar{r};q]$ then $p;r*S \subseteq r*S;q$, which is easily shown once it is seen that $r*S = (r;S)^*;\bar{r}$. Conversely, the proof that: if $p;r*S \subseteq r*S;q$ then $\exists s[p \subseteq s,\ s;r;S \subseteq r;S;s,\ s;\bar{r} \subseteq \bar{r};q]$, follows by taking $s = p\circ(r;S)^*$, and applying the properties of lemma 2.1:

1. $p = p \cap I = p\circ I \subseteq p\circ(r;S)^*$, using the definition of I, lemma 2.2.3, the definition of the * operation, and lemma 2.2.6, respectively.
2. $(p\circ(r;S)^*);r;S \subseteq r;S.(p\circ(r;S)^*)$ or (lemma 2.1)
   $(p\circ(r;S)^*)\circ(r;S) \subseteq p\circ(r;S)^*$ or (lemma 2.2.4)
   $p\circ((r;S)^*;r;S) \subseteq p\circ(r;S)^*$ or (lemma 2.2.6)
   $(r;S)^*;r;S \subseteq (r;S)^*$
   and the last inclusion follows from the definition of the * operation.
3. $(p\circ(r;S)^*);\bar{r} \subseteq \bar{r};q$ or (lemma 2.1)
   $(p\circ(r;S)^*)\circ\bar{r} \subseteq q$ or (lemma 2.2.4)
   $p\circ((r;S)^*;\bar{r}) \subseteq q$ or (def. r*S)
   $p\circ(r*S) \subseteq q$ or (lemma 2.1)
   $p;r*S \subseteq r*S;q$
   and the last inclusion follows by assumption.

In the more general case of flow diagrams, to be dealt with presently in our rephrasing of Manna's theorem, the argument is stated in somewhat more

general terms, but not essentially different. However, for the generalization to full recursion, the above mentioned extension with indexed assertions is needed.

We now first give the details of Manna's approach [12,13]. Two versions of Manna's theorem on partial correctness are given; first a weaker one, and, at the end of this section, a stronger one which is needed for the treatment of *total* correctness.

The weak version is first pictorially phrased as follows: A flow diagram P is partially correct with respect to the predicates p and q if and only if the following condition is satisfied: There exists a selection of points π_i, i=1,...,n-1, in the diagram, such that intermediate assertions $(p=p_0)\ p_1,p_2,\ldots,p_{n-1},(p_n=q)$ can be found, attached to the points π_i, for which we have that, for all i,j, $1 \leq i,j \leq n$, each $P_{i,j}$ (part of the program between π_i and π_j) is partially correct with respect to p_i and p_j, and, moreover, each part of the program is included in at least one of the $P_{i,j}$.

The formalism developed in sections 2 and 3 allows a less pictorial statement, together with a complete proof, of this theorem. We give these as preparation for the extension to programs involving full recursion, to which the remainder of the section is devoted.

We use the well-known fact that each flow diagram can be represented by an equivalent recursive program scheme such that the system of declarations (more precisely, the associated grammar (section 3)) is regular in form.

Example: Consider figure 2. This diagram may be represented by

$$(\{P_1 \Leftarrow A_1;P_2,\ P_2 \Leftarrow p_1;A_2;P_3 \cup \bar{p}_1;A_3;P_4,\ P_3 \Leftarrow p_2 \cup \bar{p}_2;A_3;P_4,$$
$$P_4 \Leftarrow p_3;P_3 \cup \bar{p}_3\},\ P_1).$$

Remark: Such translation is (first) mentioned e.g. in [17]. It is not difficult to see that the result can be obtained by the following process (only briefly sketched here):

1. Consider the flow diagram in a natural way as a finite automaton.
2. Construct the associated regular grammar.
3. Translate the grammar back into a program scheme, essentially by the operation τ^{-1} of section 3.

Using the representation of flow diagrams by regular schemes, we can now give a precise statement of our first version of Manna's theorem:

THEOREM 4.1 (Completeness theorem with regular inductive assertions). Let p,q be two predicates.
Let $\mathcal{D} = \{P_i \Leftarrow A_{i,1};P_1 \cup A_{i,2};P_2 \cup \ldots \cup A_{i,n};P_n \cup A_{i,n+1}\}_{i=1}^{n}$ be a regular declaration scheme. The program $(\mathcal{D},P_1)$ is partially correct with respect to p,q if and only if there exist $p_1,p_2,\ldots,p_{n+1}$ such that

$$\begin{cases} p \subseteq p_1 \\ \\ p_i;A_{i,j} \subseteq A_{i,j};p_j, \quad i=1,\ldots,n, \ j=1,\ldots,n+1 \\ \\ p_{n+1} \subseteq q \end{cases} \tag{4.1}$$

Remarks

1. The general form of the $\mathcal{D}$ can be specialized by taking some of the $A_{i,j}$ as I or Ω.
2. The freedom in the choice for the π_i, in P, in the flow diagram formulation is found back in the freedom of constructing $\mathcal{D}$ by, if necessary, considering subprograms of P as elementary $A_{i,j}$.

PROOF.

1. If-part. Assume (4.1). We shall prove that $p_i;P_i \subseteq P_i;p_{n+1}$, $i=1,\ldots,n$, by an application of Scott's induction rule.
   $p_i;\Omega \subseteq \Omega;p_{n+1}$ is clear. Next, we verify: If $p_i;X_i \subseteq X_i;p_{n+1}$, $i=1,\ldots,n$, then
   $$p_i;(A_{i,1};X_1 \cup \ldots \cup A_{i,n};X_n \cup A_{i,n+1}) \subseteq$$
   $$\subseteq (A_{i,1};X_1 \cup \ldots \cup A_{i,n};X_n \cup A_{i,n+1});p_{n+1}, i=1,\ldots,n.$$
   This follows from: $p_i;A_{i,j};X_j \subseteq A_{i,j};p_j;X_j \subseteq A_{i,j};X_j;p_{n+1}$, by (4.1) and the induction hypothesis, respectively. We conclude that, indeed, $p_i;P_i \subseteq P_i;p_{n+1}$. Hence, by (4.1), $p;P_1 \subseteq p_1;P_1 \subseteq P_1;p_{n+1} \subseteq P_1;q$.
2. Only-if-part. Assume $p;P_1 \subseteq P_1;q$. Two constructions for the p_j are possible.

2.1. Let $P_{n+1} \overset{df}{=} I$, and $p_j \overset{df}{=} P_j \rightarrow q, j=1,\ldots,n+1$. We verify (4.1): From

$p;P_1 \subseteq P_1;q$ we derive $p \subseteq (P_1 \to q) = p_1$. Also, $p_{n+1} = P_{n+1} \to q = I \to q = q$. In order to show that $p_i;A_{i,j} \subseteq A_{i,j};p_j$, we have to verify $(P_i \to q);A_{i,j} \subseteq A_{i,j};(P_j \to q)$, i.e.,
$\forall x,y[\forall z[xP_i z \to q(z)] \wedge xA_{i,j}y \to \forall t[yP_j t \to q(t)]]$.
Assume $\forall z[xP_i z \to q(z)]$, $xA_{i,j}y$ and $yP_j t$. Then $xA_{i,j};P_j t$, hence $xP_i t$, and $q(t)$ follows, proving (4.1).

2.2. The second construction uses the "dual" system of procedures
$\{Q_j \Leftarrow Q_1;A_{1,j} \cup \ldots \cup Q_n;A_{n,j} \cup \Delta_j\}_{j=1}^{n+1}$ with $\Delta_1 = I$, $\Delta_j = \Omega$, $j=2,\ldots,n+1$.

Example: Referring again to fig. 2, we have:
$Q_1 \Leftarrow I$, $Q_2 \Leftarrow Q_1;A_1$, $Q_3 \Leftarrow Q_2;p_1;A_2 \cup Q_4;p_3$, $Q_4 \Leftarrow Q_2;\bar{p}_1;A_3 \cup Q_3;\bar{p}_2;A_3$, $Q_5 \Leftarrow Q_3;p_2 \cup Q_4;\bar{p}_3$.
Note that the P_j denote computations from intermediate nodes in the flow diagram to the final one, whereas the Q_j denote computations from the initial to intermediate nodes.
In general, we have for the Q's: $Q_j;P_j \subseteq P_1$, and $Q_j;P_j \subseteq Q_{n+1}$, $j=1,2,\ldots,n+1$. Taking $j=n+1$ and $j=1$ in the first and second inclusion respectively, and using the definitions of P_{n+1} and Q_1, we obtain that $P_1 = Q_{n+1}$. (The proofs of these statements are omitted, since they are special cases of theorems given below.) We define $p_j = p \circ Q_j$, $j=1,2,\ldots,n+1$. The reader will have no difficulty in verifying, analogously to construction 1, that these p_j indeed satisfy (4.1). We also observe that $p \circ Q_j \subseteq P_j \to q$, $j=1,\ldots,n+1$, which again, will be proved later in a more general form. □

After thus having settled the flow diagram case (*regular* recursive schemes), we now face the problem of extending the theorem to recursive schemes in general.

Without lack of generality we assume that each declaration scheme $\mathcal{D}$ is of the form

$$\{P_i \Leftarrow S_{i,1} \cup S_{i,2} \cup \ldots S_{i,M_i}\}_{i=1}^{n} \qquad (4.2.1)$$

with M_i some integer ≥ 1, and each S_j, $j=1,\ldots,M_i$, of the form

$$S_{i,j} = A(i,j,0);P(i,j,1);\ldots \qquad (4.2.2)$$
$$\ldots;A(i,j,K_{i,j}-1);P(i,j,K_{i,j});A(i,j,K_{i,j})$$

with $A(i,j,k) \in A$, $P(i,j,k) \in \{P_1,\ldots,P_n\}$, and $K_{i,j}$ some integer ≥ 0 (if $K_{i,j}=0$, $S_{i,j}$ is just $A(i,j,0)$). Specialized forms of the $\mathcal{D}$ are again obtained by suitable restriction of certain of the $A(i,j,k)$ to I or Ω. Observe that each occurrence of some P_ℓ in some $S_{i,j}$ is uniquely identified by the triple (i,j,k) with $P(i,j,k) = P_\ell$.

A number of definitions and notations will be employed:

1. First we need a name for the set of index-triples with respect to (as will from now on be tacitly assumed) the declarations $\mathcal{D}$ as given in (4.2.1), (4.2.2):

$$\sum = \{(i,j,k) \mid 1 \leq i \leq n,\ 1 \leq j \leq M_i,\ 1 \leq k \leq K_{i,j}\}$$

2. Each $P(i,j,k)$, for $(i,j,k) \in \sum$, is some element of $\{P_1,\ldots,P_n\}$. Hence our definition of the function $h: \sum \to \{1,2,\ldots,n\}$: $h(i,j,k) = 1$ iff $P(i,j,k) = P_1$.
3. Suppression of indices will be used below to improve the clarity of the proofs. To begin with, we will use as short hand for the system $\mathcal{D} = \{P_i \Leftarrow \bigcup_{j=1}^{M_i} S_{i,j}\}_{i=1}^{n}$, with $S_{i,j}$ as above, the notation

$$P \Leftarrow A(0);P(1);\ldots;A(K-1);P(K);A(K) \qquad (4.3)$$

where both the i- and j-index have been suppressed.

An important role will be played in what follows by the idea of using index-triple sequences as *trace* of the history of the computation. We define the following subsets of $\sum^*$ (the set of *all* finite sequences of elements of $\sum$, with ε denoting the empty sequence):

$$T = T_1 \cup T_2 \cup \ldots \cup T_n$$

where the sets T_i, i=1,...,n, satisfy the system of equations

$$\{T_i = \{\varepsilon\} \cup \bigcup_{j=1}^{M_i} \bigcup_{k=1}^{K_{i,j}} (i,j,k)\, T_{h(i,j,k)}\}_{i=1,\ldots,n}$$

or, alternatively, each T_i is the language produced by the grammar $G_i = (\{T_1,\ldots,T_n\}, \sum, P_i, T_i)$, with P_i consisting of the rules $T_i \to \varepsilon$, $T_i \to (i,j,k)T_{h(i,j,k)}$, for $(i,j,k) \in \sum$.

Each T_i consists of those sequences of $\sum^*$ which satisfy

1. The first triple, if any, has i as its first index.
2. Successive triples (i,j,k), (i',j',k') are connected by the requirement that i' = h(i,j,k).

Each element $\tau_i \in T_i$ may be viewed as defining a path in the tree of incarnations of the procedures with P_i as root, or, alternatively, τ_i represents the stack of currently active procedures, each triple in τ_i representing one procedure call. This interpretation explains the requirement that i' = h(i,j,k), since i' is the index of that procedure that is located in place (i,j,k) of the scheme.

Example: Let $\mathcal{D}$ be $\{P_1 \Leftarrow A_1;P_1;A_2;P_2;A_3 \cup A_4;P_2;A_5,P_2 \Leftarrow A_6;P_1;A_7 \cup \cup A_8\}$. Then $\sum = \{(1,1,1),(1,1,2),(1,2,1),(2,1,1)\}$; also, h(1,1,1) = 1, h(1,1,2) = 2, h(1,2,1) = 2, h(2,1,1) = 1. Possible $\tau \in T$ are: ε,(1,1,1), (1,1,1)(1,1,2)(2,1,1) or (2,1,1)(1,2,1)(2,1,1), etc. The sequence τ_1 = (1,1,1)(1,1,2)(2,1,1) represents the calling structure of fig. 3.

The index-triple sequences are exploited in the introduction of the notion of *companions* of the procedures P_i: They depend on the history of the computation, represented by the index sequence τ, and come in four kinds: left-left: $L_\tau^{\lambda,i}$, left-right: $L_\tau^{\rho,i}$, right-left: $R_\tau^{\lambda,i}$, and right-right: $R_\tau^{\rho,i}$. Anticipating their precise definition, they are intended to have the following meaning: Let, for some $s \geq 0$, $\tau_{i_0} = (i_0,j_0,k_0)\ldots(i_s,j_s,k_s) \in T_{i_0} \subseteq T$, and let $i = h(i_s,j_s,k_s)$. As we saw above, τ_{i_0} keeps track of a specific path through the tree of incarnations with P_{i_0} as root, leading to the inner call of P_i. Then the computation prescribed by $L_{\tau_{i_0}}^{\lambda,i}$ is precisely the computation initiated by

the outermost call of P_{i_0}, upto, but not including, this inner call of P_i. Moreover, $L^{\rho,i}_{\tau_{i_0}} = L^{\lambda,i}_{\tau_{i_0}};P_i$. Furthermore, $R^{\rho,i}_{\tau_{i_0}}$ is the computation following after, but not including, the inner call of P_i, until completion of the outer call of P_{i_0} is achieved, and $R^{\lambda,i}_{\tau_{i_0}} = P_i;R^{\rho,i}_{\tau_{i_0}}$. Finally, $L^{\lambda,i}_{\tau_{i_0}};P_i;R^{\rho,i}_{\tau_{i_0}} \subseteq P_{i_0}$. (Compare fig. 4 and the example following the next definition.)

These notions are now defined precisely, followed by the proofs of their intended properties. Let $\mathcal{D}$ be of the form (4.3). We define two (infinite, see below) systems of procedures, one with procedure symbols $\{L^{\lambda,i}_{\tau},L^{\rho,i}_{\tau}\}_{i=1,\ldots,n,\tau\in T}$, and one with the symbols $\{R^{\lambda,i}_{\tau},R^{\rho,i}_{\tau}\}_{i=1,\ldots,n,\tau\in T}$. As abbreviation we use $L^{\lambda,h}_{\tau,k}$ instead of $L^{\lambda,h(i,j,k)}_{\tau(i,j,k)}$, and similarly for the other symbols. (It should be noted that $\tau(i,j,k)$ is the result of concatenating the index-triple sequence τ with the index-triple (i,j,k), whereas $h(i,j,k)$ is the result of applying the function h to (i,j,k).)

DEFINITION 4.1 (Companions). For each $i=1,2,\ldots,n$, $\tau \in T$:
a. (Left-companions)

$$L^{\lambda,i}_{\varepsilon} \Leftarrow I \tag{4.4.1}$$

$$L^{\lambda,h}_{\tau,1} \Leftarrow L^{\lambda,i}_{\tau};A(0) \tag{4.4.2}$$

$$L^{\lambda,h}_{\tau,k+1} \Leftarrow L^{\rho,h}_{\tau,k};A(k), \quad k=1,2,\ldots,K-1 \tag{4.4.3}$$

$$L^{\rho,i}_{\tau} \Leftarrow \bigcup_{j=1}^{M_i} \begin{cases} L^{\lambda,i}_{\tau};A(0) & , \text{ if } K_{i,j}=0 \\ L^{\rho,h}_{\tau,K};A(K) & , \text{ if } K_{i,j}>0 \end{cases} \tag{4.4.4}$$

b. (Right-companions)

$$R_{\varepsilon}^{\rho,i} \Leftarrow I \tag{4.5.1}$$

$$R_{\tau,k}^{\rho,h} \Leftarrow A(k);R_{\tau,k+1}^{\lambda,h} \qquad ,k=1,2,\ldots,K-1 \tag{4.5.2}$$

$$R_{\tau,k}^{\rho,h} \Leftarrow A(K);R_{\tau}^{\rho,i} \tag{4.5.3}$$

$$R_{\tau}^{\tau,i} \Leftarrow \bigcup_{j=1}^{M_i} \begin{cases} A(0);R_{\tau}^{\rho,i} & \text{if } K_{i,j}=0 \\ A(0);R_{\tau,1}^{\lambda,h} & \text{if } K_{i,j}>0 \end{cases} \tag{4.5.4}$$

Remark: The first appearance of *infinite* systems merits a comment: It turns out to be a straightforward matter to generalize all considerations of section 3 to infinite systems, including in particular the union and induction theorems. This is worked out in [4], but omitted here, since no special difficulties are involved.

An example of some companions: Let $\mathcal{D} = \{P \Leftarrow A_1;P;A_2;P;A_3 \cup A_4\}$. We have for the left-companions (restricting the index structure to a simpler one, as is sufficient in this example):
For $\tau \in \{0,1\}^*$: $L_{\varepsilon}^{\lambda} \Leftarrow I$, $L_{\tau 0}^{\lambda} \Leftarrow L_{\tau}^{\lambda};A_1$, $L_{\tau 1}^{\lambda} \Leftarrow L_{\tau 0}^{\rho};A_2$, $L_{\tau}^{\rho} \Leftarrow L_{\tau 1}^{\rho};A_3 \cup L_{\tau}^{\lambda};A_4$.
Hence, e.g.,

$$\begin{aligned} L_{\varepsilon}^{\rho} = L_{1}^{\rho};A_3 \cup L_{\varepsilon}^{\lambda};A_4 &= (L_{11}^{\rho};A_3 \cup L_{1}^{\lambda};A_4);A_3 \cup I;A_4 = \\ &= L_{11}^{\rho};A_3;A_3 \cup L_{0}^{\rho};A_2;A_4;A_3 \cup A_4 = \\ &= \ldots \cup (L_{01}^{\rho};A_3 \cup L_{0}^{\lambda};A_4);A_2;A_4;A_3 \cup A_4 = \\ &= \ldots \cup \ldots \cup L_{0}^{\lambda};A_4;A_2;A_4;A_3 \cup A_4 = \\ &= \ldots \cup \ldots \cup L_{\varepsilon}^{\lambda};A_1;A_4;A_2;A_4;A_3 \cup A_4 = \\ &= \ldots \cup \ldots \cup A_1;A_4;A_2;A_4;A_3 \cup A_4. \end{aligned}$$

This suggests that $L_{\varepsilon}^{\rho} = P$, which will indeed follow as one of the by-products of the first companion theorem:

THEOREM 4.2 (First companion theorem)

a. $L_{\tau}^{\lambda,i};P_i = L_{\tau}^{\rho,i}$, $i=1,\ldots,n$, $\tau \in T$

b. $P_i;R_\tau^{\rho,i} = R_\tau^{\lambda,i}$, $i=1,\ldots,n$, $\tau \in T$

PROOF. We prove only part a, part b being symmetric. Besides the system $\{L_\tau^{\lambda,i}, L_\tau^{\rho,i}\}$ we introduce - for the sake of the present proof only - the system $\{\bar{L}_\tau^{\lambda,i}, \bar{L}_\tau^{\rho,i}\}$ (the $\bar{}$ denoting an alphabetic variant, and not complementation), defined by: For $i=1,\ldots,n$, $\tau \in T$.

$$\bar{L}_\varepsilon^{\lambda,i} \Leftarrow I \tag{4.6.1}$$

$$\bar{L}_{\tau,1}^{\lambda,h} \Leftarrow L_\tau^{\lambda,i};A(0) \tag{4.6.2}$$

$$\bar{L}_{\tau,k+1}^{\lambda,h} \Leftarrow \bar{L}_{\tau,k}^{\rho,h};A(k), \quad k=1,2,\ldots,K-1 \tag{4.6.3}$$

$$\bar{L}_\tau^{\rho,i} \Leftarrow \bar{L}_\tau^{\lambda,i};P_i \tag{4.6.4}$$

We shall prove that $\{L_\tau^{\lambda,i} = \bar{L}_\tau^{\lambda,i},\ L_\tau^{\rho,i} = \bar{L}_\tau^{\rho,i}\}_{i=1,\ldots,n,\ \tau \in T}$.

Part 1. $\subseteq$. By corollary 3.1 it is sufficient to show that the $\bar{L}$ satisfy the defining inclusions of the L-system. For the $\bar{L}^\lambda$ this is immediate, since (4.4.1) to (4.4.3) are identical (apart from the $\bar{}$) to (4.6.1) to (4.6.3). For the $\bar{L}^\rho$ the proof runs as follows. We have to show:

$$\bar{L}_\tau^{\rho,i} \supseteq \bigcup_{j=1}^{M_i} \begin{cases} \bar{L}_\tau^{\lambda,i};A(0) & \text{if } K_{i,j}=0 \\ \bar{L}_{\tau,K}^{\rho,h};A(K) & \text{if } K_{i,j}>0 \end{cases} \tag{4.7}$$

If $K_{i,j}=0$, then by definition of $\mathcal{D}$, $A(0) = A(i,j,0) = S_{i,j} \subseteq P_i$. Hence, $\bar{L}_\tau^{\lambda,i};A(0) \subseteq \bar{L}_\tau^{\lambda,i};P_i = \bar{L}_\tau^{\rho,i}$, by (4.6.4). If $K_{i,j}>0$, then $\bar{L}_\tau^{\rho,i}$ $\overset{(4.6.4)}{=}$ $\bar{L}_\tau^{\lambda,i};P_i \supseteq \bar{L}_\tau^{\lambda,i};A(0);P(1);\ldots;P(K);A(K)$ $\overset{(4.6.2)}{=}$ $\bar{L}_{\tau,1}^{\lambda,h};P(1);\ldots;P(K);A(K)$ $\overset{(4.6.4)}{=}$ $\bar{L}_{\tau,1}^{\rho,h};A(1);\ldots;A(K)$ $\overset{(4.6.3)}{=}$ $\bar{L}_{\tau,2}^{\lambda,h};P(2);\ldots;A(K) = \ldots = \bar{L}_{\tau,K}^{\rho,h};A(K)$, whence (4.7) follows.

Part 2. $\supseteq$. We show that the L satisfy the defining inclusions for the $\bar{L}$. For the L^λ this is again direct from the definitions. For the L^ρ we must

show that $L_\tau^{\lambda,i};P_i \subseteq L_\tau^{\rho,i}$, for which we use Scott's induction rule on the P_i: It is sufficient to show: If $\{L_\tau^{\lambda,i};X_i \subseteq L_\tau^{\rho,i}\}_{i=1,\ldots,n,\ \tau \in T'}$ then $\{L_\tau^{\lambda,i};A(0);X(1);\ldots;X(K);A(K) \subseteq L_\tau^{\rho,i}\}_{i=1,\ldots,n,\ \tau \in T}$. If K=0, this follows from 4.4.4. If K>0, then $L_\tau^{\lambda,i};A(0);X(1);\ldots;A(K) = L_{\tau,1}^{\lambda,h};X(1);\ldots$ $\ldots;A(K) \subseteq$ (hypothesis) $L_{\tau,1}^{\rho,h};A(1);\ldots;A(K) \subseteq \ldots \subseteq L_\tau^{\rho,i}$. This completes the proof of the first companion theorem. □

COROLLARY 4.2. $L_\varepsilon^{\rho,i} = P_i$, $R_\varepsilon^{\lambda,i} = P_i$.

PROOF. Put $\tau = \varepsilon$ in theorem 4.2, and use $L_\varepsilon^{\lambda,i} = R_\varepsilon^{\rho,i} = I$. □

The next theorem combines the left- and right companions into one construct. It is, for convenience, phrased for $\tau = \tau_1 \in T_1$, but generalizes directly to indices $j \neq 1$.

THEOREM 4.3 (Second Companion theorem).

$\{L_{\tau_1}^{\lambda,i};P_i;R_{\tau_1}^{\rho,i} \subseteq P_1\}_{i=1,\ldots,n,\ \tau_1 \in T_1}$ provided that if $\tau_1 = \varepsilon$ then $i = 1$.

PROOF. Throughout the proof we require that if $\tau_1 = \varepsilon$ then $i = 1$. We shall prove the, by theorem 4.2 equivalent, inclusions

$$\left\{\begin{array}{l} L_{\tau_1}^{\lambda,i};R_{\tau_1}^{\lambda,i} \subseteq P_1 \\ L_{\tau_1}^{\rho,i};R_{\tau_1}^{\rho,i} \subseteq P_1 \end{array}\right\}_{i=1,\ldots,n,\ \tau_1 \in T_1 \text{ provided } \ldots.} \tag{4.8}$$

(4.8) is proved by (infinite) Scott induction by showing that: If

$$\left\{\begin{array}{ll} X_\varepsilon^{\lambda,1};R_\varepsilon^{\lambda,1} \subseteq P_1 & \\ X_{\tau_1,1}^{\lambda,h};R_{\tau_1,1}^{\lambda,h} \subseteq P_1 & \\ X_{\tau_1,k+1}^{\lambda,h};R_{\tau_1,k+1}^{\lambda,h} \subseteq P_1, & k=1,2,\ldots,K-1 \\ X_{\tau_1}^{\rho,i};R_{\tau_1}^{\rho,i} \subseteq P_1 & \end{array}\right.$$

then

$$\begin{cases} I;R_{\varepsilon}^{\tau,1} \subseteq P_1 \\ X_{\tau_1}^{\lambda,i};A(0);R_{\tau_1,1}^{\lambda,h} \subseteq P_1 \\ X_{\tau_1,k}^{\rho,h};A(k);R_{\tau_1,k+1}^{\lambda,h} \subseteq P_1 \quad , \ k=1,\ldots,K-1 \\ (X_{\tau_1}^{\lambda,i};A(0) \cup X_{\tau_1,K}^{\rho,h};A(K));R_{\tau_1}^{\rho,i} \subseteq P_1 \end{cases}$$

We have:

a. $I;R_{\varepsilon}^{\lambda,1} \subseteq$ (cor. 4.2) $I;P_1 \subseteq P_1$.

b. $X_{\tau_1}^{\lambda,i};A(0);R_{\tau_1,1}^{\lambda,h} \subseteq$ (4.5.4) $X_{\tau_1}^{\lambda,i};R_{\tau_1}^{\lambda,i} \subseteq P_1$, by the first three hypotheses.

c,d follow similarly by the definitions and hypotheses. □

The companion constructs are the central tool in our statement and proof of the generalized inductive assertion theorem. We use the following system of inclusions, with respect to the $\mathcal{D}$ of (4.3), and using assertions indexed in the same way as the indexed procedure letters above:

$$p_{\tau}^{i};A(0) \subseteq A(0);q_{\tau}^{i} \qquad K = 0$$

$$\left.\begin{array}{l} p_{\tau}^{i};A(0) \subseteq A(0);p_{\tau,1}^{h} \\ q_{\tau,k}^{h};A(k) \subseteq A(k);p_{\tau,k+1}^{h}, \quad k=1,\ldots,K-1 \\ q_{\tau,K}^{h};A(K) \subseteq A(K);q_{\tau}^{i} \end{array}\right\} \qquad K > 0$$

Call the system of these four inclusions $I(\mathcal{D},p_{\tau}^{i},q_{\tau}^{i})$. Then we have:

THEOREM 4.4 (Completeness theorem with generalized inductive assertions). Let p,q be two predicates. Let $\mathcal{D}$ by as in (4.3). $(\mathcal{D},P_1)$ is partially correct with respect to p and q iff there exist p_{τ}^{i}, q_{τ}^{i} such that

$$\begin{cases} p \subseteq p_\varepsilon^1 \\ q_\varepsilon^1 \subseteq q \end{cases} \quad \text{and } I(\mathcal{D}, p_{\tau_1}^i, q_{\tau_1}^i)_{i=1,\ldots,n,\ \tau_1 \in T_1}, \quad \text{if } \tau_1 = \varepsilon \text{ then } i = 1 \tag{4.9}$$

PROOF. Throughout the proof we require that if $\tau_1 = \varepsilon$ then $i = 1$.

1. If-part. Assume (4.9). We show that $p_{\tau_1}^i;P_i \subseteq P_i;q_{\tau_1}^i$. Once this has been established, the desired result follows from $p;P_1 \subseteq p_\varepsilon^1;P_1 \subseteq P_1;q_\varepsilon^1 \subseteq P_1;q$. By Scott's induction rule, it suffices to prove: If $p_{\tau_1}^i;X_i \subseteq X_i;q_{\tau_1}^i$, then $p_{\tau_1}^i;A(0);X(1);\ldots;X(K);A(K) \subseteq \subseteq A(0);X(1);\ldots;X(K);A(K);q_{\tau_1}^i$. Verification of this is direct from the definitions and the assumed inclusions in $I(\mathcal{D}, p_{\tau_1}^i, q_{\tau_1}^i)$.

2. Only-if-part. Assume $p;P_1 \subseteq P_1;q$. We have, as in theorem 4.1, two possible solutions for the $p_{\tau_1}^i, q_{\tau_1}^i$:

   First construction: $p_{\tau_1}^i \overset{df}{=} p \circ L_{\tau_1}^{\lambda,i}$, $i=1,\ldots,n$, $\tau_1 \in T_1$

   $q_{\tau_1}^i \overset{df}{=} p \circ L_{\tau_1}^{\rho,i}$, $i=1,\ldots,n$, $\tau_1 \in T_1$

   Second construction: $p_{\tau_1}^i \overset{df}{=} R_{\tau_1}^{\lambda,i} \to q$, $i=1,\ldots,n$, $\tau_1 \in T_1$

   $q_{\tau_1}^i \overset{df}{=} R_{\tau_1}^{\rho,i} \to q$, $i=1,\ldots,n$, $\tau_1 \in T_1$

   We prove only the first solution.

   a. $p_\varepsilon^1 = p \circ L_\varepsilon^{\lambda,1} = p \circ I = p$; hence, $p = p_\varepsilon^1$

   b. $q_\varepsilon^1 = p \circ L_\varepsilon^{\rho,1} = p \circ P_1$; hence, $q_\varepsilon^1 = p \circ P_1 \subseteq q$ follows.

   c. Proof of $p_{\tau_1}^i;A(0) \subseteq A(0);q_{\tau_1}^i$ (case $K = 0$). We have to show $p \circ L_{\tau_1}^{\lambda,i};A(0) \subseteq A(0);p \circ L_{\tau_1}^{\rho,i}$, which is direct from (4.4.4).

   d,e,f. The remaining cases follow from the definition of $I(\mathcal{D}, p_{\tau_1}^i, q_{\tau_1}^i)$, and (4.4.2), (4.4.3) and (4.4.4) respectively. □

COROLLARY 4.3.

1. If P_1 is partially correct with respect to p,q, then
$\{p \circ L^{\lambda,i}_{\tau_1} \subseteq R^{\lambda,i}_{\tau_1} \to q,\ p \circ L^{\rho,i}_{\tau_1} \subseteq R^{\rho,i}_{\tau_1} \to q\}_{i=1,\ldots,n,\ \tau_1 \in T_1}$
2. For each system $\{p^i_{\tau_1}, q^i_{\tau_1}\}_{i=1,\ldots,n,\ \tau_1 \in T_1}$ such that $I(\mathcal{D}, p^i_{\tau_1}, q^i_{\tau_1})$ we have $\{p^i_{\tau_1} \subseteq R^{\lambda,i}_{\tau_1} \to q,\ p \circ L^{\rho,i}_{\tau_1} \subseteq q^i_{\tau_1}\}$ $i=1,\ldots,n,\ \tau_1 \in T_1$

PROOF.

1. $p \circ L^{\lambda,i}_{\tau_1} \subseteq R^{\lambda,i}_{\tau_1} \to q$ is equivalent with $p;L^{\lambda,i}_{\tau_1};R^{\lambda,i}_{\tau_1} \subseteq L^{\lambda,i}_{\tau_1};R^{\lambda,i}_{\tau_1};q$, and this follows from $L^{\lambda,i}_{\tau_1};R^{\lambda,i}_{\tau_1} \subseteq P_1$ (theorem 4.3) and the partial correctness of P_1 with respect to p,q.
2. The technique of this proof is similar to that of the previous ones, reason why we omit it. □

One might wonder whether the complex structure of the assertions used in this proof is really needed. The following remarks show that this is indeed the case. Consider as an example the procedure P declared by $P \Leftarrow A_1;P;A_2;P;A_3 \cup A_4$. Suppose first that all partial correctness properties of P with respect to p and q could be proved already using a format with only two inductive assertions, as suggested by figure 5.

This would mean that the following formula

$$\forall p,q\Big[\text{ If } p;S \subseteq S;q \text{ then } \exists p_0,q_0 \left[\begin{array}{ll} p \subseteq p_0 & \\ & \text{and} \\ q_0 \subseteq q & \end{array} \begin{array}{l} p_0;A_1 \subseteq A_1;p_0 \\ q_0;A_2 \subseteq A_2;p_0 \\ q_0;A_3 \subseteq A_3;q_0 \\ p_0;A_4 \subseteq A_4;q_0 \end{array}\right]\Big] \tag{4.10}$$

would be true with P taken for S.

However, it can be shown that (4.10) is satisfied only by such S for which $S \supseteq (A_1 \cup A_4;A_3^*;A_2)^*;A_4;A_3^*$ holds. Thus, partial correctness properties which distinguish P from such S cannot be proved on the base of (4.10) (for an example see below).

As the next step, one might attempt to use an infinity of assertions, but with a simpler index structure than in the theorem. One might try to use

$$\forall p,q[\ \text{If } p;S \subseteq S;q \text{ then } \exists\{p_i,q_i\}_{i=0,1\ldots}\left[\begin{array}{l} p \subseteq p_0 \\ q_0 \subseteq q \end{array} \text{ and } \left\{\begin{array}{l} p_i;A_1 \subseteq A_1;p_{i+1} \\ q_{i+1};A_2 \subseteq A_2;p_{i+1} \\ q_{i+1};A_3 \subseteq A_3;q_i \\ p_i;A_4 \subseteq A_4;q_i \end{array}\right\}_{i=0,1,\ldots}\right]]$$

Then, however, it can be shown that this is satisfied only by S for which $S \supseteq \bigcap\{X \mid X = A_1;(X;A_2)^*;X;A_3 \cup A_4\}$ holds, and, again, incompleteness follows.

We now discuss an example of a property of P which is provable only with the full index structure which, due to the simple structure of P as compared with the general case, simplifies to

$$\text{If } \left\{\begin{array}{l} p_\sigma;A_1 \subseteq A_1;p_{\sigma 0} \\ q_{\sigma 0};A_2 \subseteq A_2;p_{\sigma 1} \\ q_{\sigma 1};A_3 \subseteq A_3;q_\sigma \\ p_\sigma;A_4 \subseteq A_4;q_\sigma \end{array}\right\}_{\sigma\in\{0,1\}^*} \quad \text{then } \{p_\sigma;P \subseteq P;q_\sigma\}_{\sigma\in\{0,1\}^*} \tag{4.11}$$

Take for P, in a self-explanatory notation, the special case (P manipulates states consisting of pairs of integers (n,s)):

$$P \Leftarrow [n>0;n:=n-1];P;[s:=s+1];P;[n:=n+1] \cup [n \leq 0]$$

Let p(n,s) be $(n=n_0,s=s_0)$, abbreviated to $\langle n_0,s_0\rangle$. We show that $\langle n_0,0\rangle;P \subseteq P;\langle n_0,2^{n_0}-1\rangle$, by defining p_σ,q_σ as follows: Let, for $b_i \in\{0,1\}$:

$$p_{b_1\ldots b_m} = \langle n_0-m,\ \sum_{i=1}^{m} b_i\cdot 2^{n_0-i}\rangle$$

$$q_{b_1 \ldots b_m} = \langle n_0 - m, \sum_{i=1}^{m} b_i \cdot 2^{n_0 - m} - 1 \rangle$$

Verification of (4.11) for these A_i, p_σ, q_σ is straightforward. Taking $\sigma = \varepsilon$, i.e., $m = 0$, we obtain the desired result.

The indexing strategy, though as powerful as one might expect, is not a very convenient tool for actual proofs. This may be remedied as follows (only a sketch is given): Consider once more e.g. formula (4.11). This formula may be seen as a special case of statements of the form - somewhat extending the boundaries of our formalism:

$$\text{If } \begin{cases} p(x);A_1 \subseteq A_1;p(f(x)) \\ q(f(x));A_2 \subseteq A_2;p(g(x)) \\ q(g(x));A_3 \subseteq A_3;q(x) \\ p(x);A_4 \subseteq A_4;q(x) \end{cases} \quad \text{then } p(x);P \subseteq P;q(x)$$

where p(x) and q(x) are parametrized properties of the - suitably defined - state, and f and g transform the parameter (in the special case of (4.11), one could take $f(\sigma) = \sigma 0$, $g(\sigma) = \sigma 1$). It follows immediately that this proof technique has the same power as the rigid formalism of theorem 4.4 (which is the minimal one needs to obtain completeness). E.g., in the example given above, we can define

$$p(x,y)(n,s) = \langle x,y \rangle$$
$$q(x,y)(n,s) = \langle x, y+2^x-1 \rangle$$
$$f(x,y) = (x-1,y)$$
$$g(x,y) = (x-1, y+2^{x-1})$$

which is much more natural and allows a simpler verification.

We now continue with the application of theorem 4.4 to obtain an alternative for the minimal fixed point characterization of recursive procedures:

COROLLARY 4.4. Let $\mathcal{D}$ be as before, and let $R_1, \ldots, R_n$ be arbitrary statements. Then

$$\forall \ell = 1,\ldots,n, p^{(\ell)}, q^{(\ell)} \Big[p^{(\ell)}; R_\ell \subseteq R_\ell; q^{(\ell)} \text{ iff } \exists \{p^i_{\tau_\ell}, q^i_{\tau_\ell}\}_{i=1,\ldots,n,\ \tau_\ell \in T_\ell}$$

$$\text{such that} \begin{cases} p^{(\ell)} \subseteq p^\ell_\varepsilon \\ q^\ell_\varepsilon \subseteq q^{(\ell)} \end{cases} \text{and } I(\mathcal{D}, p^i_{\tau_\ell}, q^i_{\tau_\ell})_{i=1,\ldots,n, \tau_\ell \in T_\ell} \Big]$$

iff

$\forall \ell = 1,\ldots,n\ [R_\ell = P_\ell]$.

PROOF. Follows from theorem 4.4 and lemma 2.6. □

We conclude our paper with a discussion of the notion of *total* correctness and its relationship to partial correctness.

P is totally correct with respect to q iff $\forall x\, \exists y[xPy \wedge q(y)]$. In order to explain the relationship with partial correctness, we once more consider the simple while statement r*S. In the beginning of this section we saw that r*S is partially correct with respect to p and q iff there exists s such that $p \subseteq s$, $s;r;S \subseteq r;S;s$, and $s;\bar{r} \subseteq \bar{r};q$, i.e.,

$$\begin{aligned} &\forall x,y\ [p(x) \wedge x\ r^*S\ y \rightarrow q(y)] \\ &\leftrightarrow \qquad\qquad (4.12) \\ &\exists s[\forall x[p(x) \rightarrow s(x)] \wedge \forall y,z[s(y) \wedge r(y) \wedge ySz \rightarrow s(z)] \wedge \\ &\qquad\qquad \wedge \forall t[s(t) \wedge r(t) \rightarrow q(t)]] \end{aligned}$$

We are interested in particular in the case that p is identically true. Suppose we could prove, for such p, the following stronger version of (4.12):

$$\begin{aligned} &\forall x\ [\forall y[x\ r^*S\ y \rightarrow q(y)] \\ &\quad \leftrightarrow \qquad\qquad (4.13) \\ &\quad \exists s[s(x) \wedge \forall y,z[s(y) \wedge r(y) \wedge ySz \rightarrow s(z)] \wedge \\ &\qquad\qquad \forall t[s(t) \wedge r(t) \rightarrow q(t)]]]. \end{aligned}$$

From this we may conclude, by replacing q by $\neg q$, and negating both sides:

$$\forall x\,[\neg\forall y[x\ r{*}S\ y \rightarrow \neg q(y)] \leftrightarrow \neg\exists s[s(x) \wedge \forall y,z[s(y) \wedge r(y) \wedge ySz \rightarrow s(z)] \wedge \forall t[s(t) \wedge r(t) \rightarrow \neg q(t)]]].$$

Now observe that $\neg\forall y[x\ r{*}S\ y \rightarrow \neg q(y)] \leftrightarrow \exists y[\ x\ r{*}S\ y \wedge q(y)]$, i.e., $r{*}S$ is totally correct in x with respect to q. Thus we see that if we could prove (4.13) then, writing $\neg\exists s E(x,s,\neg q)$ for its right-hand side, we could justify the inference of total correctness of $r{*}S$ in x with respect to q from the proof of $\neg\exists s E(x,s,\neg q)$, i.e., from the negation of partial correctness (in the refined sense) of $r{*}S$ in x with respect to (the identically true p and) $\neg q$. This inference seems to be the essence of Manna's treatment of total correctness.

We therefore will prove an extension of the generalized inductive assertion theorem, yielding the equivalent of (4.13) in the general case:

THEOREM 4.5 (Total correctness)

$$\forall x[\ \forall y[xP_1y \rightarrow q(y)] \leftrightarrow \exists p^i_{\tau_1}, q^i_{\tau_1} \left[\begin{array}{l} p^1_\varepsilon(x) \\ \forall t[q^1_\varepsilon(t) \rightarrow q(t)] \end{array} \text{ and } \mathcal{I}(\mathcal{D}, p^i_{\tau_1}, q^i_{\tau_1})_{i=1,\ldots,n,\ \tau_1 \in T_1 \text{ if } \tau_1 = \varepsilon \text{ then } i = 1}\right]]$$

PROOF. We give only the $\rightarrow$ part. Choose some fixed x_0, and let $p^i_{\tau_1}, q^i_{\tau_1}$ be defined by

$$p^i_{\tau_1} \overset{df}{=} \{(x_0,x_0)\} \circ L^{\lambda,i}_{\tau_1}$$

$$q^i_{\tau_1} \overset{df}{=} \{(x_0,x_0)\} \circ L^{\rho,i}_{\tau_1}$$

(Note that $\{(x_0,x_0)\} \subseteq I$ is indeed an assertion.)

We show that

1. $p_\varepsilon^1(x_0)$ holds: $p_\varepsilon^1(x_0) = (\{(x_0,x_0)\}\circ L_\varepsilon^{\lambda,1})(x_0) = (\{(x_0,x_0)\}\circ I)(x_0) = \{(x_0,x_0)\}(x_0)$, and $\{(x_0,x_0)\}(x_0)$ is clearly satisfied.
2. $\forall t[q_\varepsilon^1(t) \to q(t)]$, i.e., $\forall t[(\{(x_0,x_0)\}\circ L_\varepsilon^{\rho,1})(t) \to q(t)]$, or $\forall t[\exists y\{(x_0,x_0)\}(y) \wedge yP_1t \to q(t)]$, or $\forall t,y[y=x_0 \wedge yP_1t \to q(t)]$, or $\forall t[x_0 P_1 t \to q(t)]$, which holds by assumption.
3. The proof that $I(\mathcal{D},p_{\tau_1}^i,q_{\tau_1}^i)$ holds is similar to that of theorem 4.4, and omitted. □

With this last theorem we hope to have clarified the precise status of the notion of total correctness, thus achieving the last goal of our paper.

REFERENCES

[1] Ashcroft, E.A., Z. Manna & A. Pnueli, Decidable properties of monadic functional schemas, J. ACM, 20 (1973) 489-499.

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

[3] De Bakker, J.W. & W.P. de Roever, A calculus for recursive program schemes, in Automata, Languages and Programming (M. Nivat,ed.), p. 167-196, North-Holland, Amsterdam, (1973).

[4] De Bakker, J.W. & L.G.L. T. Meertens, Simple recursive program schemes and inductive assertions, Report MR 142, Mathematisch Centrum, Amsterdam, (1972).

[5] Dijkstra, E.W., A simple axiomatic basis for programming language constructs, to appear.

[6] Engelfriet, J., Recursion induction and Floyd's method, memorandum 25, Twente Technical University, Enschede, (1971).

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

[8] Ginsburg, S., The Mathematical Theory of Context Free Languages, McGraw-Hill, New York, (1966).

[9] Hitchcock, P. & D.M.R. Park, Induction rules and proofs of termination, in Automata, Languages and Programming (M. Nivat, ed.), p. 225 - 251, North-Holland, Amsterdam, (1973).

[10] Hoare, C.A.R., An axiomatic basis for computer programming, C. ACM, 12 (1969) 576-580.

[11] Kleene, S.C., Introduction to Metamathematics, North-Holland, Amsterdam, (1952).

[12] Manna, Z., The correctness of programs, J. CSS, 3 (1969) 119-127.

[13] Manna, Z., Mathematical theory of partial correctness, in Symposium on Semantics of Algorithmic Languages (E. Engeler, ed.),

p. 252-269, Lecture Notes in Mathematics, Vol. 188, Springer, Berlin (1971).

[14] Manna, Z., S. Ness & J. Vuillemin, Inductive methods for proving properties of programs, C. ACM, 16 (1973) 491-502.

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

[16] Manna, Z. & J. Vuillemin, Fixpoint approach to the theory of computation, C. ACM, 15 (1972) 528-536.

[17] McCarthy, J., Towards a mathematical science of computation, in Proc. IFIP Congress 1962, p. 21-28, North-Holland, Amsterdam, (1963).

[18] Milner, R., An approach to the semantics of parallel programs, to appear.

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

[20] Turing, A.M., On checking a large routine, in Report of a Conference on High-Speed Automatic Calculating Machines, p. 67-69, University Mathematical Laboratory, Cambridge (1949).

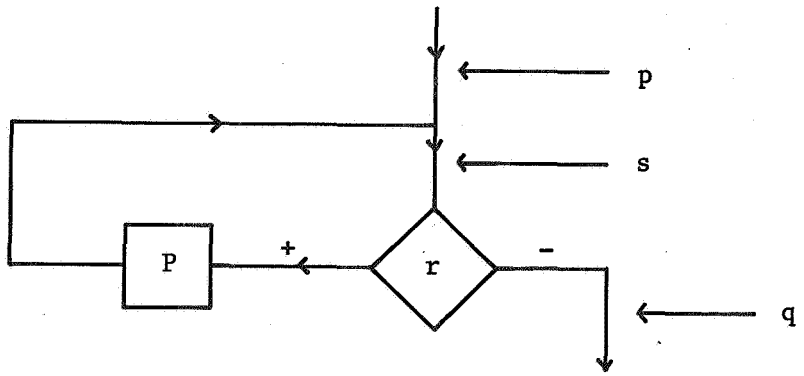

Figure 1. The inductive assertion method for the while statement r * P

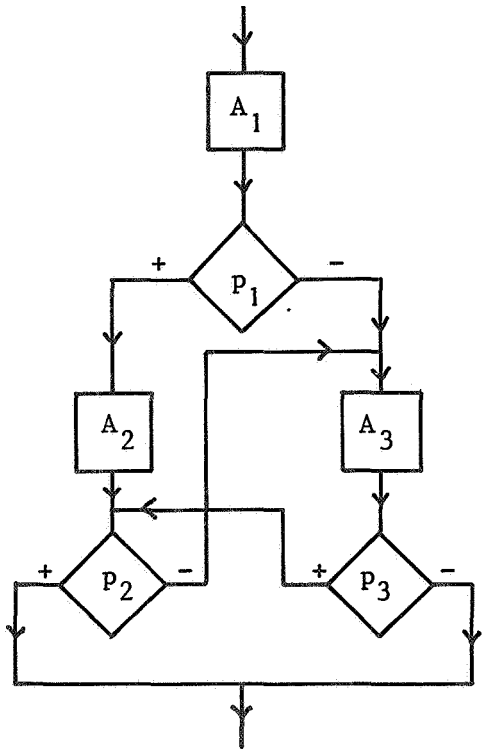

Figure 2. Example of a flow diagram represented by a set of (regular) procedure declarations

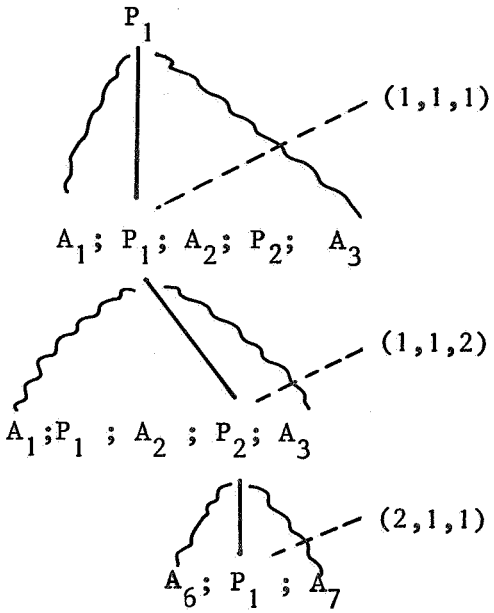

Figure 3. A tree of incarnations of recursive procedures with associated index-triple sequence

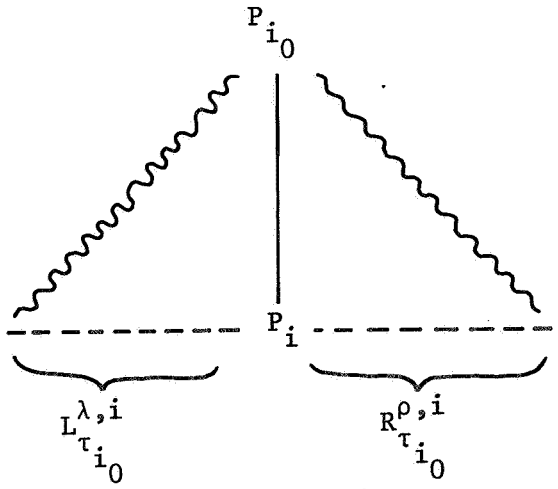

Figure 4. Left- and right companions of P_i in a tree with root P_{i_0}

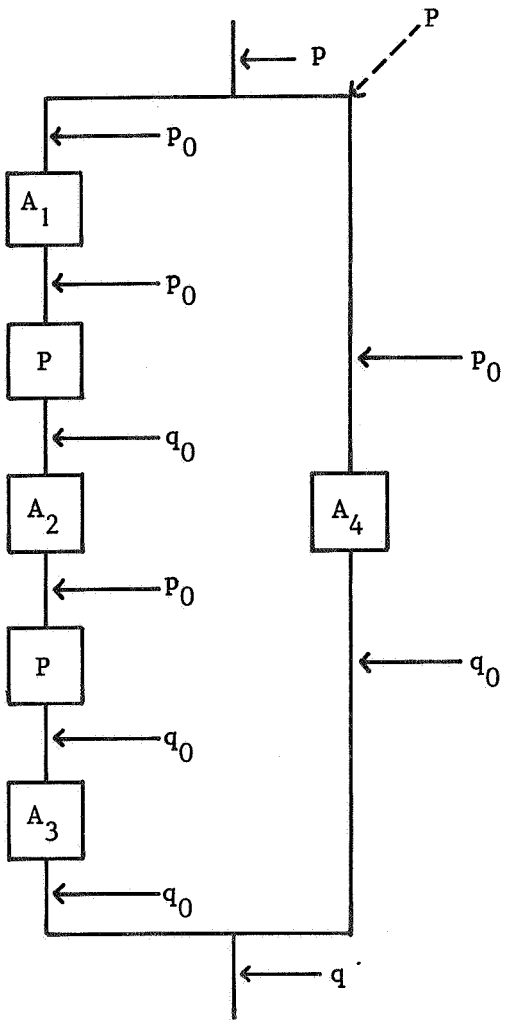

Figure 5. An incomplete system of only two intermediate inductive assertions