Reprinted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
All Rights Reserved by Academic Press, New York and London

Vol. 11, No. 3, December 1975
Printed in Belgium

On the Completeness of the Inductive Assertion Method*

J. W. DE BAKKER AND L. G. L. T. MEERTENS

Mathematical Centre, 2e Boerhaavestraat 49, Amsterdam-1005, The Netherlands

Received January 31, 1974

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.

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 [6], 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: 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., [11, 12]), 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 [11, 12] were not worked out, and, moreover, the theorem was restricted to programs in flow diagram form.

* This publication is registered as Report IW 12/73 of the Mathematical Centre.

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 [14] 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 for stating and proving 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 $P \to p$). A number of basic properties of these operations are derived, and a few remarks on other aspects of the relational approach are made.

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 [18]), are derived again. However, we chose a different approach from, e.g., that of [3] by this time exploiting 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., [7]) to the "languages" of elementary actions defined by procedures.

Section 4 brings the main result of the paper. First, the completeness theorem for the flow chart case is proved by way of introduction. Use is made of the well-known technique of replacing the flow chart by an equivalent system of recursive procedures which are *regular* in form; i.e., each term contains at most one procedure call, and this is the last operation in the term. A finite system of intermediate assertions, one for each procedure in the system, suffices here. Next, the general case is treated, viz, of a system of declarations in *context-free* form. This time an infinite system of assertions is needed. The main step in their construction is a technique for associating with a finite context-free system an equivalent infinite regular system. Once this is done, the intermediate assertions are obtained in the same way as with the flow chart case. The formalism of the just-mentioned construction is rather forbiddingly complex. However, it is shown both that a simpler construction will not work and that there is a way of looking at the construction which does lead to practical applications (Section 5). 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 induction 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. Second, the relationship between the notion of correctness given above (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. However, in Section 5 we illustrate by means of an example—the recursive solution to the Towers of Hanoi puzzle—that our technique does have practical applications.

As related work, besides the already mentioned paper, we should note that of Engelfriet [5], who is also concerned with completeness results for flow diagrams.

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

The present paper is a modification and extension of our 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, at the start, nondeterministic 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, (nondeterministic) 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 is this analysis of the state 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 ($n \geqslant 0$). 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:

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

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 at elementary programs as we have above. It suffices to have "elementary actions" $A_1, A_2, \ldots$, 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 asignment 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 $\mathscr{V}$ be the domain of states, and let R, R_1, $R_2, \ldots$, be binary relations over $\mathscr{V}$ (i.e., subsets of $\mathscr{V} \times \mathscr{V}$). Then we define

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

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

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

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

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: $p_+ = \{(x, x) \mid p(x) \text{ is true}\}$, $p_- = \{(x, x) \mid p(x) \text{ is false}\}$. 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). *In the sequel, all predicates are assumed to be total.* 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 $q; q = q$, for each q which is a subset of 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 $R * p =^{\mathrm{df}} R; p * R$ (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 go-to 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. For a possible objection to this definition, compare the remark made below when we introduce the relational formulation of termination of programs.

Unless explicitly stated contrarily, we shall from now on *identify* programs with their associated relations. A possible objection is that, occasionally, we shall need *two* equality relations between programs P_1 and P_2, viz, syntactic identity stating that the two symbol strings P_1 and P_2 are identical, and, second, semantic identity stating that the relations (associated with) P_1 and P_2 coincide (i.e., this is the equivalence relation just introduced). Normally, we shall mean the second equality relation, and we reserve the symbol "$=$" for this. In the few cases where we want to express syntactic equality, we shall do so by using the symbol "$\equiv$."

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

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

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 will usually 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.

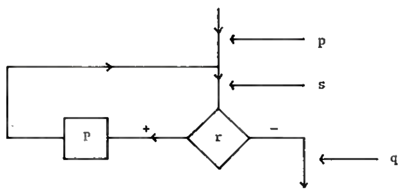

FIG. 1. The inductive assertion method for the *while* statement $r * P$.

According to the Floyd technique (which, in essence, was proposed earlier by Turing, in [19]; 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 is Manna's partial correctness theorem [11, 12] in its simplest form. 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) [9]. Using this notation, he introduces various *axioms*. For example, 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 $(\cdots {}^{X}_{a} \cdots)X := f(X)(\cdots {}^{X}_{f(a)} \cdots)$ 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 make two remarks.

First, note that both p; $P \subseteq P$; q and P; $q \subseteq p$; P are, like many more correctness statements, all special forms of a $P \subseteq \underset{\sim}{Q}$ inclusion (e.g., for the first take $x\underset{\sim}{Q}y \leftrightarrow [p(x) \rightarrow q(y)]$), 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. [17]). 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, $I \subseteq P$; $\check{P}$, 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 $\underset{\sim}{Q}$ 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 $P \cup \underset{\sim}{Q} = P$ ("$\cup$" taken as programming construct denoting nondeterministic choice), since the left-hand side may, if the second alternative is chosen, end 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 [8]; we will not pursue these problems further here.

Now, back to correctness. We once more consider 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:

$$\forall y[\exists x[p(x) \wedge xPy] \rightarrow q(y)], \qquad \text{and} \qquad \forall x[p(x) \rightarrow \forall y[xPy \rightarrow q(y)]].$$

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 the identically 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) $p \circ P = \bigcap \{q \mid p; P \subseteq P; q\}$, $P \rightarrow q = \bigcup \{p \mid p; P \subseteq P; q\}$.

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 $p \circ \check{P}$ and $\check{P} \rightarrow q$, for which we have $p \circ \check{P} = \bigcap \{q \mid p; \check{P} \subseteq \check{P}; q\} = \bigcap \{q \mid P; p \subseteq q; P\}$, and $\check{P} \rightarrow q = \bigcup \{p \mid P; p \subseteq q; P\}$. (Observe that here we used $P \subseteq Q \leftrightarrow \check{P} \subseteq \check{Q}$, $(P_1 ; P_2)^{\smile} = \check{P}_2 ; \check{P}_1$, and $\check{p} = p$ for $p \subseteq I$.)

The basic properties of the "$\circ$" and "$\rightarrow$" operations are collected in Lemmas 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* $\check{P}$ *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. $\forall x, y[xP; I \circ Py \leftrightarrow xPy \wedge (I \circ P)(y) \leftrightarrow xPy \wedge \exists z[I(z) \wedge zPy]$
$\leftrightarrow xPy \wedge \exists z[zPy] \leftrightarrow xPy]$.

4. $\forall x[(p \circ (P_1 ; P_2))(x) \leftrightarrow \exists y[p(y) \wedge yP_1 ; P_2x]$
$$\leftrightarrow \exists y, z[p(y) \wedge yP_1z \wedge zP_2x]$$
$$\leftrightarrow \exists z[\exists y[p(y) \wedge yP_1z] \wedge zP_2x]$$
$$\leftrightarrow \exists z[(p \circ P_1)(z) \wedge zP_2x] \leftrightarrow ((p \circ P_1) \circ P_2)(x)].$$ ▮

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

LEMMA 2.3. (1) $P \rightarrow I = I, \quad I \rightarrow p = p.$

(2) $I \subseteq (p_1 \rightarrow p_2)$ *iff* $p_1 \subseteq p_2$.

(3) $(P \rightarrow \Omega); P = \Omega.$

(4) $(P_1 ; P_2) \rightarrow p = (P_1 \rightarrow (P_2 \rightarrow p)).$

(5) $(P_1 \cup P_2) \rightarrow p = (P_1 \rightarrow p) \cap (P_2 \rightarrow p).$

Proof. Immediate. ▮

When we compare $p \circ \check{P} = \bigcap \{q \mid P; p \subseteq q; P\}$, and $P \rightarrow p = \bigcup \{q \mid q; P \subseteq P; p\}$, 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 $\check{P}; P \subseteq I$, or, equivalently, $\forall x, y, z[xPy \wedge xPz \rightarrow y = z]$. P is total iff $I \subseteq P; \check{P}$ or, equivalently, $\forall x \exists y[xPy]$. We then have:

LEMMA 2.4. (1) *If P is a function, then $p \circ \check{P} \subseteq P \rightarrow p$.*

(2) *If, for all p, $p \circ \check{P} \subseteq P \rightarrow p$, then P is a function.*

(3) *If P is total, then $P \rightarrow p \subseteq p \circ \check{P}$.*

(4) *If, for all p, $P \rightarrow p \subseteq p \circ \check{P}$, then P is total.*

(5) *(Conclusion) P is a total function iff $\forall p[P \rightarrow p = p \circ \check{P}]$.*

Proof. *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. ▮

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 $p \subseteq I$, $\bar{p} =^{\mathrm{df}} I \backslash p$.

LEMMA 2.5. (1) $p \circ P = U; p; P \cap I$. *(Remember that U is the universal relation.)*

(2) $P \rightarrow p = \overline{\bar{p} \circ \check{P}}.$

Proof. Left to the reader. ∎

As the 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 $\underset{\sim}{Q}$, R_1 is correct with respect to $\underset{\sim}{Q}$ ($R_1 \subseteq \underset{\sim}{Q}$) iff R_2 is correct with respect to $\underset{\sim}{Q}$, 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 lemmas: Let $R^{\dagger} =^{\mathrm{df}} (I \circ \check{R}) * R$; 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 already knows about procedures taken as minimal fixed points and Scott's induction and 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., [7]; this result may be seen as an instance of Kleene's first recursion theorem [10]).

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 the procedure symbols are now added, by means of composition and union (where the last construct is used to model conditionals).

More formally, a (recursive) program T consists of a set of declarations

$$\mathscr{D} = \{P_1 \Leftarrow S_1, P_2 \Leftarrow S_2, ..., P_n \Leftarrow S_n\},$$

and a statement S; i.e., $T = (\mathscr{D}, S)$. Here "$\Leftarrow$" stands for "is recursively defined by" (in ALGOL 60 we would write *procedure* P_i ; S_i , $i = 1, 2, ..., n$). Observe that $\mathscr{D}$ 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, $i = 1, ..., n$, or S, may contain occurrences of the P_i, $i = 1, ..., n$, and we write $S_i \equiv S_i(P_1, P_2, ..., P_n)$, $S \equiv S(P_1, P_2, ..., P_n)$. 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 $S(S^{(1)}, S^{(2)}, ..., S^{(n)})$.

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, we can only remark, first, that there *is* a correspondence (given below) between parameterless procedures and the *monadic* recursive function schemes of, e.g., [1], and, second, that it will appear, we hope, that even parameterless procedures lead to interesting considerations which, moreover, are needed anyway for a full understanding of procedures *with* parameters. So much for the apology.

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

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

(1) $\mathscr{R} \cup \mathscr{P} \subseteq \mathscr{S}(\mathscr{R}, \mathscr{P})$;

(2) if $S_1, S_2 \in \mathscr{S}(\mathscr{R}, \mathscr{P})$, then $(S_1 ; S_2)$ and $(S_1 \cup S_2) \in \mathscr{S}(\mathscr{R}, \mathscr{P})$.

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, so that we can later *prove* that we may drop them unambiguously.

Our task now is to find the relations corresponding to procedures, just as we did before for the constructs of sequencing, selection, and simple iteration. As before, we assume known how the elementary actions are executed, and now we must 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 $\mathscr{S}(\mathscr{R}, \mathscr{P})$.

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 $T = (\mathscr{D}, S)$, when applied to initial state x, follows rather naturally: Begin with initial computation point (I, x, S) ($S_l \equiv I$: 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 with some final (S', y, I), with S' some sequence of elementary actions, y the final state, and $S_r \equiv I$ indicating that nothing remains to be done.

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

DEFINITION 3.2. Let $\mathscr{D}$ be a set of declarations. A computation step is a *$\mathscr{D}$-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) $S_r \equiv (R; S_r')$, for some $R \in \mathscr{R}$, and, moreover, xRx', and $S_l' \equiv S_l; R$ hold. (Observe that this implies that $R \not\equiv \Omega$, and that, if $R \equiv p \in \mathscr{B}$, then xpx', or, equivalently, $x = x'$ and $p(x)$ hold.)

(1b) $S_r \equiv R$, for some $R \in \mathscr{R}$, $S_r' \equiv I$, and, moreover, xRx' and $S_l' \equiv S_l ; R$ hold.

(2a) $S_r \equiv ((S_{r,1} ; S_{r,2}); S_{r,3})$, $\quad S_r' \equiv (S_{r,1} ; (S_{r,2} ; S_{r,3}))$,
$S_l' \equiv S_l$, $\quad x' = x$.

(2b) $S_r \equiv (S_{r,1} \cup S_{r,2})$, $\quad S_r' \equiv S_{r,1}$ or $S_r' \equiv S_{r,2}$,
$S_l' \equiv S_l$, $\quad x' = x$.

(2c) $S_r' \equiv ((S_{r,1} \cup S_{r,2}); S_{r,3})$, $\quad S_r' \equiv ((S_{r,1} ; S_{r,3}) \cup (S_{r,2} ; S_{r,3}))$,
$S_l' \equiv S_l$, $\quad x' = x$.

(2d) $S_r \equiv (P; S_{r,1})$, $\quad S_r' \equiv S; S_{r,1}$, where $P \Leftarrow S \in \mathscr{D}$,
$S_l' \equiv S_l$, $\quad x' = x$.

(Observe that the replacement of the procedure identifier P by its body S is the usual copy rule of procedure semantics.)

(2e) $S_r \equiv P$, $\quad S_r' \equiv S$, where $P \Leftarrow S \in \mathscr{D}$,
$S_l' \equiv S_l$, $\quad x' = x$.

EXAMPLE (not bothering for the moment about parentheses). A sequence of $\mathscr{D}$-allowed computation steps, where $\mathscr{D}$ 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 $T = (\mathscr{D}, S)$ should give no problem:

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

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

From Definition 3.3, a number of properties follow rather directly, which is 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) *Similarly for left-distributivity of* ";" *over* "$\cup$".

(5) *If* $P \Leftarrow S \in \mathscr{D}$, *then* $P = S$. (*This is the fixed-point property of procedures.*)

(6) $xS_1 ; S_2y$ (*according to Definition* 3.3) *iff* $\exists z[xS_1z \wedge zS_2y]$, *and* $xS_1 \cup S_2y$ (*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* $\mathscr{D}$*-allowed computation step, then* $S_l ; S_r \supseteq S_l'; S_r'$.

(9) $S = \bigcup \{S' \mid \exists x, y$ *such that* $(I, x, S),\ldots, (S', y, I)$ *is a sequence of* $\mathscr{D}$*-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 $(\mathscr{P}, \mathscr{R}, \mathscr{D}, S)$ reminds one of a context-free grammar, with $\mathscr{P}$: nonterminals; $\mathscr{R}$: terminals; $\mathscr{D}$: productions rules; and S: start symbol.

2. The way in which the $\mathscr{D}$-allowed computation steps are defined—in particular, the procedure–call step (2d, 2e)—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 a procedure P declared by $P \Leftarrow p; A_1 ; P; A_2 \cup \bar{p}$. Suppose we choose p, A_1, and A_2 such that we have as instances of P, in a self-explanatory notation, $P \Leftarrow [x > 0 \mid x := x - 1]; P; [x := x + 1] \cup [x = 0]$. Our assertion that $P = \bigcup_{n=0}^{\infty} ((x := x - 1)^n; x := 0; (x := x + 1)^n)$ will not be surprising, nor the similarity of this expression with the "language" $\{(x := x - 1)^n; x := 0; (x := x + 1)^n \mid n \geqslant 0\}$. We now make these informal observations more precise.

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

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

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

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

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

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

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

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

The next definition introduces the language associated with a program.

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

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

where $\overset{*}{\Rightarrow}_{\tau(T)}$ 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 $\mathscr{L}(\tau(T)) = \{(pA)^i \bar{p} \mid i \geqslant 0\}$.

So far everything has been 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 would coincide with the language of Definition 3.4. There is a slight complication, however. For example, $T = (\{P \Leftarrow p; A_1 \cup \bar{p}; A_2\}, p; P)$. Then $\mathscr{L}(\tau(T)) = (ppA_1, p\bar{p}A_2)$, but there is no x, y such that $(I, x, p; P), ..., (p; \bar{p}; A_2, y, I)$ is an allowed sequence of computation steps.

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

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

Proof. Direct from the definitions. ∎

Continuing with the last example, $\mathscr{L}(\tau(T)) = \{ppA_1, p\bar{p}A_2\}$. Hence,

$$\tau^{-1}(\mathscr{L}(\tau(T))) = p; p; A_1 \cup p; \bar{p}; A_2 = p; p; A_1 \cup \Omega = p; p; A_1 = T.$$

We have now reached the point where we can apply the result of, e.g., [7], which states that context-free languages are minimal solutions of a system of equations the solutions for which are obtainable by means of successive approximations, starting from the empty set.

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

$$\tau(S)^{[0]} \overset{\text{df}}{=} \varnothing,$$

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

Then, by [7], $\mathscr{L}(\tau(T)) = \bigcup_{j=0}^{\infty} \tau(S)^{[j]}$. Hence by Lemma 3.2,

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

Now let $S^{(0)} =^{\text{df}} \Omega$, $S^{(j+1)} =^{\text{df}} S(S_1^{(j)}, ..., S_n^{(j)})$. Then it is not difficult to verify that $S^{(j)} = \tau^{-1}(\tau(S)^{[j]})$, and, moreover,

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

Thus, we have

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

With reference to $\mathscr{D}$ once more omitted this yields

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

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

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

$$S_i^{(0)} = \Omega \subseteq Q_i\,,$$
$$S_i^{(j+1)} = S_i(S_1^{(j)},..., S_n^{(j)}) \subseteq S_i(Q_1\,,..., Q_n) \subseteq Q_i$$

(by monotonicity (Lemma 3.1.7) and the induction hypothesis). Thus, $\bigcup_{j=0}^{\infty} S_i^{(j)} \subseteq Q_i$, whence $P_i \subseteq Q_i\,, i = 1,..., n$, follows. ∎

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

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

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

(1) (*Correctness in terms of inclusion.*)

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

(2) (*Characterizing recursion in terms of inclusion correctness.*) $\forall R_1\,,..., R_n\,,$

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

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, 13, 15]).

THEOREM 3.2. (Scott's induction rule). *Let* $\mathscr{D} = \{P_i \Leftarrow S_i\}_{i=1}^{n}$. *Let*

$$S_l \equiv S_l(P_1\,,..., P_n) \qquad \textit{and} \qquad S_r \equiv S_r(P_1\,,..., P_n)$$

be two statements satisfying the two conditions:

(1) $S_l(\Omega,..., \Omega) \subseteq S_r(\Omega,..., \Omega)$.

(2) *If* $S_l(X_1\,,..., X_n) \subseteq S_r(X_1\,,..., X_n)$, *then* $S_l(S_1(X_1\,,..., X_n),..., S_n(X_1\,,..., X_n)) \subseteq S_r(S_1(X_1\,,..., X_n),..., S_n(X_1\,,..., X_n))$.

Then we have that $S_l \equiv S_l(P_1\,,..., P_n) \subseteq S_r(P_1\,,..., P_n) \equiv S_r$.

Proof. As before, for $S \in \mathcal{S}(\mathcal{R}, \mathcal{P})$, let $S^{(0)} = \Omega$, $S^{(j+1)} = S(S_1^{(j)},..., S_n^{(j)})$. By condition 1, we see that $S_l^{(1)} \subseteq S_r^{(1)}$. Then, using condition 2 (with Ω for X_i), we infer that $S_l(S_1(\Omega,..., \Omega),..., S_n(\Omega,..., \Omega)) \subseteq S_r(S_1(\Omega,..., \Omega),..., S_n(\Omega,..., \Omega))$, i.e., that $S_l^{(2)} \subseteq S_r^{(2)}$. Repeating this argument we obtain that $S_l^{(j)} \subseteq S_r^{(j)}$, $j = 0, 1, 2,...$, and the desired conclusion $S_l = \bigcup_{j=0}^{\infty} S_l^{(j)} \subseteq \bigcup_{j=0}^{\infty} S_r^{(j)} = S_r$ 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 $S_l \subseteq S_r$.

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.

To bring out the difficulty, we once more consider formulas (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, by Lemma 2.1,
$(p \circ (r; S)^*) \circ (r; S) \subseteq p \circ (r; S)^*$, or, by Lemma 2.2.4,
$p \circ ((r; S)^*; r; S) \subseteq p \circ (r; S)^*$, or, by 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, by Lemma 2.1,
$(p \circ (r; S)^*) \circ \bar{r} \subseteq q$, or, by Lemma 2.2.4,
$p \circ ((r; S)^*; \bar{r}) \subseteq q$, or, by Definition $r * S$,
$p \circ (r * S) \subseteq q$; or, by 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 differently. However, for the generalization to full recursion, the above-mentioned extension with indexed assertions is needed.

We first give the details of Manna's approach. 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 $(p = p_0)\, p_1, p_2, ..., p_{n-1}, (p_n = q)$ can be found, attached to the points π_i, for which we have that, for all i, j, $1 \leqslant i, j \leqslant n$, 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 Fig. 2. This diagram may be represented by $(\{P_1 \Leftarrow A_1 ; P_2,\ P_2 \Leftarrow p_1 ; A_2 ; P_3 \cup \bar{p}_1 ; A_3 ; P_4,\ P_3 \Leftarrow p_2 \cup \bar{p}_2 ; A_3 ; P_4,\ P_4 \Leftarrow p_3 ; P_3 \cup \bar{p}_3\}, P_1)$.

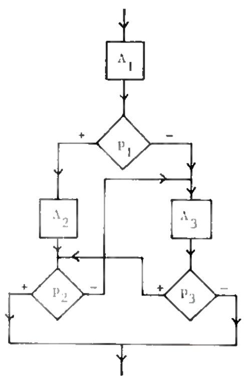

FIG. 2. Example of a flow diagram represented by a set of (regular) procedure declarations.

Remark. Such translation is (first) mentioned, e.g., in [16]. 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 $\mathscr{D} = \{P_i \Leftarrow A_{i,1} ; P_1 \cup A_{i,2} ; P_2 \cup \cdots \cup A_{i,n} ; P_n \cup A_{i,n+1}\}_{i=1}^{n}$ be a regular declaration scheme. The program $(\mathscr{D}, P_1)$ is partially correct with respect to p, q if and only if there exist $p_1, p_2, \ldots, p_{n+1}$ such that*

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

Remarks. (1) The general form of the $\mathscr{D}$ 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 $\mathscr{D}$ 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 $p_i ; (A_{i,1} ; X_1 \cup \cdots \cup A_{i,n} ; X_n \cup A_{i,n+1}) \subseteq (A_{i,1} ; X_1 \cup \cdots \cup A_{i,n} ; X_n \cup A_{i,n+1}); p_{n+1}$, $i = 1,\ldots, n$. 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 $P_{n+1} =^{\mathrm{df}} I$, and $p_j =^{\mathrm{df}} P_j \to q$, $j = 1,\ldots, n+1$. We verify (4.1):

From $p; P_1 \subseteq P_1 ; q$ we derive $p \subseteq (P_1 \to q) = p_1$. Also,

$$p_{n+1} = P_{n+1} \to q = I \to q = q.$$

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

$$\{Q_j \Leftarrow Q_1\,;\,A_{1,j} \cup \cdots \cup Q_n\,;\,A_{n,j} \cup \Delta_j\}_{j=1}^{n+1} \text{ with } \Delta_1 = I, \quad \Delta_j = \Omega, \quad j = 2,\ldots, n+1.$$

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 inclusions, 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 $\mathscr{D}$ is of the form

$$\{P_i \Leftarrow S_{i,1} \cup S_{i,2} \cup \cdots S_{i,M_i}\}_{i=1}^{n}, \tag{4.2.1}$$

with M_i some integer $\geqslant 1$, and each $S_{i,j}$, $j = 1,\ldots, M_i$, of the form

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

with $A(i,j,k) \in \mathscr{A}$, $P(i,j,k) \in \{P_1,\ldots, P_n\}$, and $K_{i,j}$ some integer $\geqslant 0$ (if $K_{i,j} = 0$, $S_{i,j}$ is just $A(i,j,0)$). Specialized forms of the $\mathscr{D}$ 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 $P(i,j,k) = P_l$.

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 $\mathscr{D}$ as given in (4.2.1) and (4.2.2):

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

2. Each $P(i,j,k)$, for $(i,j,k) \in \Sigma$, is some element of $\{P_1,\ldots, P_n\}$. Hence our definition of the function h: $\Sigma \to \{1, 2,\ldots, n\}$: $h(i,j,k) = l$ iff $P(i,j,k) = P_l$.

3. Suppression of indices will be used below to improve the clarity of the proofs. To begin with, we will use as shorthand for the system

$$\mathscr{D} = \left\{ P_i \Leftarrow \bigcup_{j=1}^{M_i} S_{i,j} \right\}_{i=1}^{n},$$

with $S_{i,j}$ as above, the notation

$$P \Leftarrow A(0); P(1); \ldots ; A(K-1); P(K); A(K), \tag{4.3}$$

where both the i- and the 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 Σ^* (the set of *all* finite sequences of elements of Σ, with ϵ denoting the empty sequence):

$$T = T_1 \cup T_2 \cup \cdots \cup T_n,$$

where the sets T_i, $i = 1,\ldots, n$, satisfy the system of equations

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

or, alternatively, each T_i is the language produced by the grammar

$$G_i = (\{T_1, \ldots, T_n\}, \Sigma, R_i, T_i),$$

with R_i consisting of the rules $T_i \to \epsilon$, $T_i \to (i, j, k)\, T_{h(i,j,k)}$, for $(i, j, k) \in \Sigma$.

Each T_i consists of those sequences of Σ^* 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 $\mathscr{D}$ be $\{P_1 \Leftarrow A_1 ; P_1 ; A_2 ; P_2 ; A_3 \cup A_4 ; P_2 ; A_5 , P_2 \Leftarrow A_6 ; P_1 ; A_7 \cup A_8\}$. Then $\Sigma = \{(1, 1, 1), (1, 1, 2), (1, 2, 1), (2, 1, 1)\}$; 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 $\tau_1 = (1, 1, 1)(1, 1, 2)(2, 1, 1)$ represents the calling structure of Fig. 3.

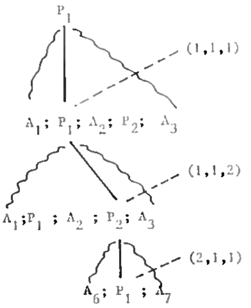

FIG. 3. A tree of incarnations of recursive procedures with associated index-triple sequence.

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 meaning: For some $s \geqslant 0$, let

$$\tau_{i_0} = (i_0\,, j_0\,, k_0) \cdots (i_s\,, j_s\,, k_s) \in T_{i_0} \subseteq T,$$

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}, up to, but not including, this inner call of P_i. Moreover, $L_{\tau_{i_0}}^{\rho,i} = L_{\tau_{i_0}}^{\lambda,i}$; P_i. Furthermore, $R_{\tau_{i_0}}^{\rho,i}$ 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 $R_{\tau_{i_0}}^{\lambda,i} = P_i$; $R_{\tau_{i_0}}^{\rho,i}$. Finally, $L_{\tau_{i_0}}^{\lambda,i}$; P_i ; $R_{\tau_{i_0}}^{\rho,i} \subseteq P_{i_0}$. (Compare Fig. 4 and the example following the next definition.)

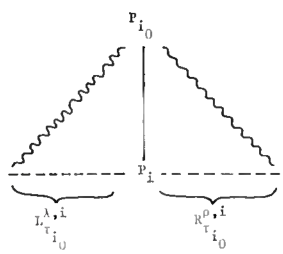

FIG. 4. Left and right companions of P_i in a tree with root P_{i_0}.

These notions are now defined precisely, followed by the proofs of their intended properties. Let $\mathscr{D}$ be of the form (4.3). We define two (infinite; see below) systems of procedures, one with procedure symbols $\{L_\tau^{\lambda,i}, L_\tau^{\rho,i}\}_{i=1,\ldots,n,\tau\in T}$, and one with the symbols $\{R_\tau^{\lambda,i}, R_\tau^{\rho,i}\}_{i=1,\ldots,n,\tau\in T}$. As abbreviation we use $L_{\tau,k}^{\lambda,h}$ instead of $L_{\tau(i,j,k)}^{\lambda,h(i,j,k)}$, 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)

$$L_\epsilon^{\lambda,i} \Leftarrow I, \tag{4.4.1}$$

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

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

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

(b) (Right companions)

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

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

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

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

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 $\mathscr{D} = \{P \Leftarrow A_1;\ P;\ A_2;\ P;\ A_3 \cup A_4\}$. We have for the left companions (restricting the index structure to a simpler one, as is sufficient in this example):

For $\tau \in \{0, 1\}^*$: $L_\epsilon^\lambda \Leftarrow I$, $L_{\tau 0}^\lambda \Leftarrow L_\tau^\lambda;\ A_1$, $L_{\tau 1}^\lambda \Leftarrow L_{\tau 0}^\rho;\ A_2$, $L_\tau^\rho \Leftarrow L_{\tau 1}^\rho;\ A_3 \cup L_\tau^\lambda;\ A_4$.
Hence, e.g.,

$$\begin{aligned} L_\epsilon^\rho = L_1^\rho;\ A_3 \cup L_\epsilon^\lambda;\ A_4 &= (L_{11}^\rho;\ A_3 \cup L_1^\lambda;\ A_4);\ A_3 \cup I;\ A_4 \\ &= L_{11}^\rho;\ A_3;\ A_3 \cup L_0^\rho;\ A_2;\ A_4;\ A_3 \cup A_4 \\ &= \cdots \cup (L_{01}^\rho;\ A_3 \cup L_0^\lambda;\ A_4);\ A_2;\ A_4;\ A_3 \cup A_4 \\ &= \cdots \cup \cdots \cup L_0^\lambda;\ A_4;\ A_2;\ A_4;\ A_3 \cup A_4 \\ &= \cdots \cup \cdots \cup L_\epsilon^\lambda;\ A_1;\ A_4;\ A_2;\ A_4;\ A_3 \cup A_4 \\ &= \cdots \cup \cdots \cup A_1;\ A_4;\ A_2;\ A_4;\ A_3 \cup A_4. \end{aligned}$$

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

THEOREM 4.2 (First companion theorem).

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

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

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 $^-$ denoting an alphabetic variant, and not complementation), defined by: For $i = 1,\ldots, n$, $\tau \in T$,

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

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

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

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

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)–(4.4.3) are identical (apart from the $^-$) to (4.6.1)–(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 $\mathscr{D}$, $A(0) = A(i,j,0) = S_{i,j} \subseteq P_i$. 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 $\bar{L}_\tau^{\rho,i}$ by (4.6.4) $= \bar{L}_\tau^{\lambda,i}; P_i \supseteq \bar{L}_\tau^{\lambda,i}$; $A(0); P(1);\ldots; P(K); A(K)$ by (4.6.2) $= \bar{L}_{\tau,1}^{\lambda,h}; P(1);\ldots; P(K); A(K)$ by (4.6.4) $= \bar{L}_{\tau,1}^{\rho,h}$; $A(1);\ldots; A(K)$ by (4.6.3) $= \bar{L}_{\tau,2}^{\lambda,h}; P(2);\ldots; A(K) = \cdots = \bar{L}_{\tau,K}^{\rho,h}; A(K)$, 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 $\{L_\tau^{\lambda,i}; A(0); X(1);...; X(K); A(K) \subseteq L_\tau^{\rho,i}\}_{i=1,...,n,\tau\in T}$. If $K = 0$, this follows from (4.4.4). If $K > 0$, then $L_\tau^{\lambda,i}; A(0); X(1);...; A(K) = L_{\tau,1}^{\lambda,h}; X(1);...; A(K) \subseteq$ (hypothesis) $L_{\tau,1}^{\rho,h}; A(1);...; A(K) \subseteq \cdots \subseteq L_\tau^{\rho,i}$. This completes the proof of the first companion theorem. ▮

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

Proof. Put $\tau = \epsilon$ in Theorem 4.2, and use $L_\epsilon^{\lambda,i} = R_\epsilon^{\rho,i} = I$. ▮

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).

$$\{L_{\tau_1}^{\lambda,i}; P_i; R_{\tau_1}^{\rho,i} \subseteq P_1\}_{i=1,...,n\ \tau_1\in T_1}$$

provided that if $\tau_1 = \epsilon$ *then* $i = 1$.

Proof. Throughout the proof we require that if $\tau_1 = \epsilon$ then $i = 1$. We shall prove the, by Theorem 4.2 equivalent, inclusions

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

We prove (4.8) by (infinite) Scott induction by showing that: If

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

then

$$\left\{\begin{array}{ll} I; R_\epsilon^{\lambda,1} \subseteq P_1, & \\ X_{\tau_1}^{\lambda,i}; A(0); R_{\tau_1,1}^{\lambda,h} \subseteq P_1, & \\ X_{\tau_1,k}^{\rho,h}; A(k); R_{\tau_1,k+1}^{\lambda,h} \subseteq P_1, & k = 1,..., K-1, \\ (X_{\tau_1}^{\lambda,i}; A(0) \cup X_{\tau_1,K}^{\rho,h}; A(K)); R_{\tau_1}^{\rho,i} \subseteq P_1. & \end{array}\right.$$

We have (a) $I;\ R_\epsilon^{\lambda,1} \subseteq$ (Corollary 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) and (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 $\mathscr{D}$ of (4.3), and using assertions indexed in the same way as the indexed procedure letters above:

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

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

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

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

$$\begin{cases} p \subseteq p_\epsilon^1, \\ q_\epsilon^1 \subseteq q, \end{cases} \quad \textit{and} \quad \mathscr{I}(\mathscr{D}, p_{\tau_1}^i, q_{\tau_1}^i)_{\substack{i=1,\dots,n,\ \tau_1 \in T_1 \\ \text{if } \tau_1 = \epsilon, \text{ then } i=1}}. \tag{4.9}$$

Proof. Throughout the proof we require that if $\tau_1 = \epsilon$, 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 $p;\ P_1 \subseteq p_\epsilon^1;\ P_1 \subseteq P_1;\ q_\epsilon^1 \subseteq P_1;\ q$. By Scott's induction rule, it suffices to prove: If $p_{\tau_1}^i;\ X_i \subseteq X_i;\ q_{\tau_1}^i$, then

$$p_{\tau_1}^i;\ A(0);\ X(1);\dots;\ X(K);\ A(K) \subseteq A(0);\ X(1);\dots;\ X(K);\ A(K);\ q_{\tau_1}^i.$$

Verification of this is direct from the definitions and the assumed inclusions in $\mathscr{I}(\mathscr{D}, p_{\tau_1}^i, q_{\tau_1}^i)$.

(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:

$$p_{\tau_1}^i \overset{\text{df}}{=} p \circ L_{\tau_1}^{\lambda,i}, \qquad i = 1,\dots, n, \quad \tau_1 \in T_1,$$

$$q_{\tau_1}^i \overset{\text{df}}{=} p \circ L_{\tau_1}^{\rho,i}, \qquad i = 1,\dots, n, \quad \tau_1 \in T_1.$$

Second construction:

$$p^i_{\tau_1} \overset{\text{df}}{=} R^{\lambda,i}_{\tau_1} \to q, \qquad i = 1,\ldots, n, \quad \tau_1 \in T_1\,,$$

$$q^i_{\tau_1} \overset{\text{df}}{=} R^{\rho,i}_{\tau_1} \to q, \qquad i = 1,\ldots, n, \quad \tau_1 \in T_1\,.$$

We prove only the first solution.

(a) $p^1_\epsilon = p \circ L^{\lambda,1}_\epsilon = p \circ I = p$; hence, $p = p^1_\epsilon$.

(b) $q^1_\epsilon = p \circ L^{\rho,1}_\epsilon = p \circ P_1$; hence, $q^1_\epsilon = p \circ P_1 \subseteq q$ follows.

(c) Proof of $p^i_{\tau_1}$; $A(0) \subseteq A(0)$; $q^i_{\tau_1}$ (case $K = 0$): We have to show that $p \circ L^{\lambda,i}_{\tau_1}$; $A(0) \subseteq A(0)$; $p \circ L^{\rho,i}_{\tau_1}$, which is direct from (4.4.4).

(d), (e), (f) The remaining cases follow from the definition of $\mathscr{I}(\mathscr{D}, p^i_{\tau_1}, q^i_{\tau_1})$, 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 $\mathscr{I}(\mathscr{D}, p^i_{\tau_1}, q^i_{\tau_1})$, 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, which is 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 could be proved using a format with only two inductive assertions, as suggested by Fig. 5a.

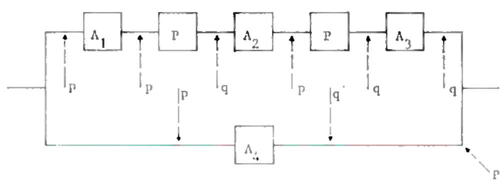

FIG. 5a. A finite, and incomplete, system of intermediate assertions.

We shall use $I_0(p, q, A_{1-4})$ as the name for the system of four inclusions $p; A_1 \subseteq A_1$; $p, q; A_2 \subseteq A_2 ; p, q; A_3 \subseteq A_3 ; q$, and $p; A_4 \subseteq A_4 ; q$. We now argue as follows. Consider the two sets

$$M_1 = \{S: \forall p, q[\text{if } I_0(p, q, A_{1-4}) \text{ then } p; S \subseteq S; q]\},$$
$$M_2 = \{S: S \subseteq (A_1 \cup A_4 ; A_3^*; A_2)^*; A_4 ; A_3^*\}.$$

It can be verified that $M_1 = M_2$. Now let p_0, q_0 be a pair of assertions such that $p_0 ; P \subseteq P; q_0$, but, for some $S_0 \in M_2$, *not* $p_0 ; S_0 \subseteq S_0 ; q_0$. Clearly, such p_0, q_0, S_0 always exist. Then, since $M_1 = M_2$, $S_0 \in M_1$, and we see that $I_0(p_0, q_0, A_{1-4})$ does not hold, since, otherwise, $p_0 ; S_0 \subseteq S_0 ; q_0$ could be inferred. Thus, we conclude that $p_0 ; P \subseteq P; q_0$ is a partial correctness property which is not provable with the simple structure as in I_0.

Next, let us consider the infinite, but not sufficiently refined, system of inductive assertions as suggested by Fig. 5b. We then argue essentially as above, with obvious changes in M_1, and changing M_2 to $\{S: S \subseteq \bigcap \{X: X = A_1 ; (X; A_2)^*; X; A_3 \cup A_4\}\}$. The details of this case are left to this reader.

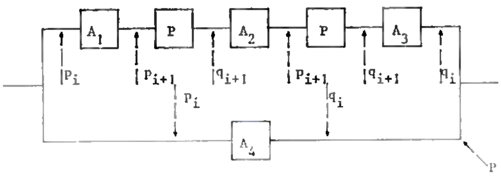

Fig. 5b. An infinite, but still incomplete, system of intermediate assertions.

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 $\mathscr{D}$ be as before, and let $R_1, \ldots, R_n$ be arbitrary statements. Then*

$$\forall l = 1,\ldots, n, p^{(l)}, q^{(l)} \Big[p^{(l)}; R_l \subseteq R_l ; q^{(l)} \quad \textit{iff} \quad \exists\{p^i_{\tau_l}, q^i_{\tau_l}\}_{i=1,\ldots,n,\ \tau_l \in T_l}$$
$$\textit{such that} \quad \begin{cases} p^{(l)} \subseteq p_\epsilon^{\ l}, \\ q_\epsilon^{\ l} \subseteq q^{(l)}, \end{cases} \quad \textit{and} \quad \mathscr{I}(\mathscr{D}, p^i_{\tau_l}, q^i_{\tau_l})_{i=1,\ldots,n,\ \tau_l\in T_l} \Big]$$

iff $\forall l = 1,\ldots, n\ [R_l = P_l]$.

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* $\forall x\, \exists y[xPy \wedge q(y)]$. 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.,

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

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 (4.10):

$$\begin{aligned} &\forall x[\forall y[x\ r * S\ y \rightarrow q(y)] \\ &\quad \leftrightarrow \exists s[s(x) \wedge \forall y, z[s(y) \wedge r(y) \wedge ySz \rightarrow s(z)] \\ &\qquad \wedge \forall t[s(t) \wedge \neg r(t) \rightarrow q(t)]]]. \end{aligned} \tag{4.11}$$

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

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

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 (4.11), then, writing $\neg \exists s \mathscr{E}(x, s, \neg q)$ 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 $\neg \exists s \mathscr{E}(x, s, \neg q)$, 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 (4.11) in the general case:

THEOREM 4.5 (Total correctness).

$$\forall x \left[\forall y[xP_1 y \rightarrow q(y)] \right.$$
$$\left. \leftrightarrow \exists p^i_{\tau_1}, q^i_{\tau_1} \begin{bmatrix} p^1_\epsilon(x), \\ \forall t[q^1_\epsilon(t) \rightarrow q(t)], \end{bmatrix} \quad \textit{and} \quad \mathscr{I}(\mathscr{D}, p^i_{\tau_1}, q^i_{\tau_1})_{\substack{i=1,\ldots,n,\ \tau_1 \in T_1 \\ \text{if } \tau_1=\epsilon, \text{ then } i=1}} . \right]$$

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

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

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

(1) $p_\epsilon^1(x_0)$ holds:

$$p_\epsilon^1(x_0) = (\{(x_0, x_0)\} \circ L_\epsilon^{\lambda,1})(x_0) = (\{(x_0, x_0)\} \circ I)(x_0) = \{(x_0, x_0)\}(x_0),$$

and $\{(x_0, x_0)\}(x_0)$ is clearly satisfied.

(2) $\forall t[q_\epsilon^1(t) \rightarrow q(t)]$, i.e., $\forall t[(\{(x_0, x_0)\} \circ L_\epsilon^{\rho,1})(t) \rightarrow q(t)]$, or

$$\forall t[\exists y\{(x_0, x_0)\}(y) \wedge yP_1t \rightarrow q(t)],$$

or $\forall t, y[y = x_0 \wedge yP_1t \rightarrow q(t)]$, or $\forall t[x_0P_1t \rightarrow q(t)]$, which holds by assumption.

(3) The proof that $\mathscr{I}(\mathscr{D}, p_{\tau_1}^i, q_{\tau_1}^i)$ holds is similar to that of Theorem 4.4, and is 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.

5. An Application

After having obtained—in the form of the completeness theorem—the main result of our paper, we now indicate a way of applying this result in a proof of program correctness. We shall prove the correctness of the wellknown recursive solution of the Towers of Hanoi problem. A proof based directly on the indexing mechanism of Section 4 might be possible, but it would be very awkward. Instead, we reformulate Theorem 4.4 in such a way that practical application becomes feasible. We shall not do so in full generality, but restrict ourselves to the problem at hand, leaving the formulation of the general case to the reader.

Remember that the Towers of Hanoi puzzle is concerned with the following. There are three piles of disks, with, initially, N disks at pile 1, say, positioned in such a way that each disk is smaller than the disk below it. The problem is to move the N disks from pile 1 to pile 3, say, where pile 2 may be used for temporary storage, in such a way that the constraint that a disk be smaller than the disk below it is obeyed, at each of the three piles, at all intermediate positions (including, of course, the final one).

Let f (*from*), v (*via*), t (*to*) be three variables with distinct values in $\{1, 2, 3\}$, let n be an integer $\geqslant 0$, and let move $(n + 1, f, t)$ be the elementary action of moving disk $n + 1$ from pile f to pile t. The recursive solution of our problem is then given by the procedure P declared as follows (the notation, insofar as it is not yet introduced, should be self explanatory).

$$\begin{aligned}P \Leftarrow\ & [n > 0 \mid n := n - 1; (f, v, t) := (f, t, v)];\\ & P;\\ & [(f, v, t) := (f, t, v); \text{move}(n + 1, f, t); (f, v, t) := (v, f, t)];\\ & P;\\ & [(f, v, t) := (v, f, t); n := n + 1]\\ & \cup\\ & [n = 0].\end{aligned} \tag{5.1}$$

Observe that P has the format of

$$P \Leftarrow A_1 ; P; A_2 ; P; A_3 \cup A_4 .$$

For P in this form, Theorem 4.4 simplifies to

$$\forall p, q \left[\begin{array}{l} p; P \subseteq P; q \\ \text{iff} \\ \exists \{p_\sigma , q_\sigma\}_{\sigma \in \{0,1\}^*} \text{ such that } p \subseteq p_\epsilon , q_\epsilon \subseteq q, \text{ and} \\ \left\{\begin{array}{l} p_\sigma ; A_1 \subseteq A_1 ; p_{\sigma 0} \\ q_{\sigma 0} ; A_2 \subseteq A_2 ; p_{\sigma 1} \\ q_{\sigma 1} ; A_3 \subseteq A_3 ; q_\sigma \\ p_\sigma ; A_4 \subseteq A_4 ; q_\sigma \end{array}\right\}_{\sigma \in \{0,1\}^*} \end{array}\right]. \tag{5.2}$$

The generalization of this referred to above is the following. We observe that in (5.2) we are concerned with a set $\mathscr{V} = \{0, 1\}^*$, and operations $f, g\colon \mathscr{V} \to \mathscr{V}$ defined by $f(\sigma) = \sigma 0$, $g(\sigma) = \sigma 1$. Now the particular structure of $\mathscr{V}$, f, and g plays no role in the proof of the if part of (5.2). In fact, we readily see that (5.2) can be restated as

$$\forall p, q \left[\begin{array}{l} p; P \subseteq P; q \\ \text{iff} \\ \exists \mathscr{V}, f\colon \mathscr{V} \to \mathscr{V}, g\colon \mathscr{V} \to \mathscr{V}, \{p(\sigma), q(\sigma)\}_{\sigma \in \mathscr{V}} , \sigma_0 \in \mathscr{V} \\ \text{such that } p \subseteq p(\sigma_0), q(\sigma_0) \subseteq q, \\ \text{and} \\ \left\{\begin{array}{l} p(\sigma); A_1 \subseteq A_1 ; p(f(\sigma)) \\ q(f(\sigma)); A_2 \subseteq A_2 ; p(g(\sigma)) \\ q(g(\sigma)); A_3 \subseteq A_3 ; q(\sigma) \\ p(\sigma); A_4 \subseteq A_4 ; q(\sigma) \end{array}\right\}_{\sigma \in \mathscr{V}} \end{array}\right]. \tag{5.3}$$

This formulation gives us the key to the proof of the correctness of (5.1). First we define the structure of the states on which the procedure P of (5.1) operates. Each state x is a 5-tuple

$$x = (n, f, v, t, (d_1 ,..., d_N)),$$

with n an integer, f, v, t as above, and $(d_1, \ldots, d_N)$ an N-tuple of variables with values in $\{1, 2, 3\}$. If d_j has the value i, $1 \leqslant i \leqslant 3$, this indicates that in state x disk j is located at pile i. To be more specific, let us consider the case in which $N = 7$ and we are at a certain intermediate state $x = (4, 3, 2, 1, (3, 3, 3, 3, 2, 1, 3))$. To this state we then have Fig. 6 as the corresponding picture, and this picture describes the situation in which we have to perform the subtask of moving the four disks on top of pile 3 to pile 1 (via pile 2).

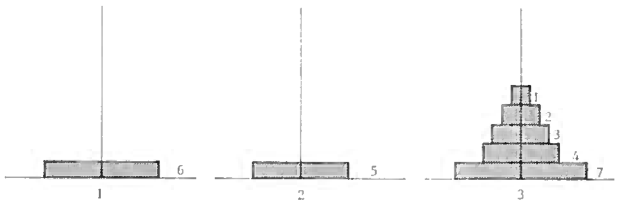

FIG. 6. Intermediate state $x = (4, 3, 2, 1, (3, 3, 3, 3, 2, 1, 3))$.

Next, we describe the structure of the parameter σ which occurs in the assertions $p(\sigma)$, $q(\sigma)$. Each σ is a 5-tuple

$$\sigma = (\nu, \phi, \beta, \tau, (\delta_{\nu+1}, \ldots, \delta_N)).$$

Its meaning is explained after the definitions of the $p(\sigma)$, $q(\sigma)$, which are as follows.

$$p(\sigma)(x) = \left\{\begin{matrix} n = \nu, \quad f = \phi, \quad v = \beta, \quad t = \tau, \\ d_j = \begin{cases} \phi, & 1 \leqslant j \leqslant \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant n, \end{cases} \end{matrix}\right\},$$

$$q(\sigma)(x) = \left\{\begin{matrix} n = \nu, \quad f = \phi, \quad v = \beta, \quad t = \tau, \\ d_j = \begin{cases} \tau, & 1 \leqslant j \leqslant \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant n, \end{cases} \end{matrix}\right\}.$$

These definitions are to be interpreted as follows. For state x and parameter σ, $p(\sigma)(x)$ and $q(\sigma)(x)$ are true before, resp. after, performing the subtask of moving the n uppermost disks from pile f to pile t: p and q differ only in the conditions imposed upon the d_j, $1 \leqslant j \leqslant \nu$. For these j, if n has current value ν, then $d_j = \phi$ (current value of f), before, and $d_j = \tau$ (current value of t), after performing the subtask. All other variables are unchanged, and their current values are stored in the parameter σ.

Next we define the functions f and g:

$$f(\sigma) = (\nu - 1, \phi, \tau, \beta, (\phi, \delta_{\nu+1}, \ldots, \delta_N)),$$
$$g(\sigma) = (\nu - 1, \beta, \phi, \tau, (\tau, \delta_{\nu+1}, \ldots, \delta_N)).$$

Finally, σ_0 is defined as

$$\sigma_0 = (N, 1, 2, 3, (\ )).$$

Note that, according to the definition of p, q, and σ_0, we then have

$$p(\sigma_0)(x) = \{n = N, f = 1, v = 2, t = 3, d_j = 1\ (1 \leqslant j \leqslant N)\},$$
$$q(\sigma_0)(x) = \{n = N, f = 1, v = 2, t = 3, d_j = 3\ (1 \leqslant j \leqslant N)\},$$

and our aim is to use (5.3) in order to show that if $p(\sigma_0)(x)$ holds before execution of P, then $q(\sigma_0)(x)$ holds after its execution, thus establishing the correctness of P. We have to verify the four conditions $p(\sigma)$; $A_1 \subseteq A_1$; $p(f(\sigma)),\ldots, p(\sigma)$; $A_4 \subseteq A_4$; $q(\sigma)$, with the definitions of the p, q, σ, x, $A_1,\ldots, A_4$ filled in. Below we write out the first three inclusions, omitting the trivial case of the fourth, and writing $\{p\}\ S\{q\}$ as shorthand for p; $S \subseteq S$; q.

(1) $$\left\{n = \nu,\ f = \phi,\ v = \beta,\ t = \tau,\ d_j = \begin{cases}\phi, & 1 \leqslant j \leqslant \nu \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\right\},$$

$[n > 0 \mid n := n - 1; (f, v, t) := (f, t, v)]$,

$$\left\{n = \nu - 1,\ f = \phi,\ v = \tau,\ t = \beta,\ d_j = \begin{cases}\phi, & 1 \leqslant j \leqslant \nu - 1, \\ \begin{cases}\phi, & j = \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\end{cases}\right\}.$$

(2) $$\left\{n = \nu - 1,\ f = \phi,\ v = \tau,\ t = \beta,\ d_j = \begin{cases}\beta, & 1 \leqslant j \leqslant \nu - 1, \\ \begin{cases}\phi, & j = \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\end{cases}\right\},$$

$[(f, v, t) := (f, t, v);\ \text{move}(n + 1, f, t);\ (f, v, t) := (v, f, t)]$,

$$\left\{n = \nu - 1, f = \beta, v = \phi, t = \tau, d_j = \begin{cases}\beta, & 1 \leqslant j \leqslant \nu - 1, \\ \begin{cases}\tau, & j = \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\end{cases}\right\}.$$

(3) $$\left\{n = \nu - 1, f = \beta, v = \phi, t = \tau, d_j = \begin{cases}\tau, & 1 \leqslant j \leqslant \nu - 1, \\ \begin{cases}\tau, & j = \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\end{cases}\right\},$$

$[(f, v, t) := (v, f, t);\ n := n + 1]$,

$$\left\{n = \nu, f = \phi, v = \beta, t = \tau, d_j = \begin{cases}\tau, & 1 \leqslant j \leqslant \nu, \\ \delta_j, & \nu + 1 \leqslant j \leqslant N,\end{cases}\right\}.$$

The reader will have no problem in verifying that inclusions 1–3 are satisfied. Thus, we can indeed apply (5.3). Adding to this the observation that the formalism excludes both illegal intermediate positions and illegal moves, we have proved the correctness of the Towers of Hanoi program.

REFERENCES

1. E. A. ASHCROFT, Z. MANNA, AND A. PNUELI, Decidable properties of monadic functional schemas, *J. Assoc. Comput. Mach.* **20** (1973), 489–499.
2. J. W. DE BAKKER, "Recursive Procedures," Mathematical Centre Tracts 24, Mathematisch Centrum, Amsterdam, 1971.
3. J. W. DE BAKKER AND W. P. DE ROEVER, A calculus for recursive program schemes, *in* "Automata, Languages and Programming" (M. Nivat, Ed.), pp. 167–196, North-Holland, Amsterdam, 1973.
4. J. W. DE BAKKER AND L. G. L. T. MEERTENS, Simple recursive program schemes and inductive assertions, Report MR 142, Mathematisch Centrum, Amsterdam, 1972.
5. J. ENGELFRIET, Recursion induction and Floyd's method, Memorandum 25, Twente Technical University, Enschede, 1971.
6. R. W. FLOYD, Assigning meanings to programs, *in* "Mathematical Aspects of Computer Science" (J. T. Schwartz, Ed.), pp. 19–32, Proceedings of a Symposium in Applied Mathematics, Vol. 19, American Math. Soc., Providence, 1967.
7. S. GINSBURG, "The Mathematical Theory of Context Free Languages," McGraw-Hill, New York, 1966.
8. P. HITCHCOCK AND D. M. R. PARK, Induction rules and proofs of termination, *in* "Automata, Languages and Programming" (M. Nivat, Ed.), pp. 225–251, North-Holland, Amsterdam, 1973.
9. C. A. R. HOARE, An axiomatic basis for computer programming, *Comm. Assoc. Comput. Mach.* **12** (1969), 576–580.
10. S. C. KLEENE, "Introduction to Metamathematics," North-Holland, Amsterdam, 1952.
11. Z. MANNA, The correctness of programs, *J. Comput. System Sci.* **3** (1969), 119–127.
12. Z. MANNA, Mathematical theory of partial correctness, *in* "Symposium on Semantics of Algorithmic Languages" (E. Engeler, Ed.), pp. 252–269, Lecture Notes in Mathematics, Vol. 188, Springer, Berlin, 1971.
13. Z. MANNA, S. NESS, AND J. VUILLEMIN, Inductive methods for proving properties of programs, *Comm. Assoc. Comput. Mach.* **16** (1973), 491–502.
14. Z. MANNA AND A. PNUELI, Formalization of properties of functional programs, *J. Assoc. Comput. Mach.* **17** (1970), 555–569.
15. Z. MANNA AND J. VUILLEMIN, Fixpoint approach to the theory of computation, *Comm. Assoc. Comput. Mach.* **15** (1972), 528–536.
16. J. MCCARTHY, Towards a mathematical science of computation, *in* "Proceedings of the IFIP Congress 1962," pp. 21–28, North-Holland, Amsterdam, 1963.
17. R. MILNER, An approach to the semantics of parallel programs, to appear.
18. D. SCOTT AND J. W. DE BAKKER, A theory of programs: Notes of an IBM Vienna seminar, unpublished, 1969.
19. A. M. TURING, On checking a large routine, *in* Report of a Conference on High-Speed Automatic Calculating Machines, pp. 67–69, University Mathematical Laboratory, Cambridge, 1949.

Printed by the St Catherine Press Ltd., Tempelhof 37, Bruges, Belgium.