stichting
mathematisch
centrum

MC

REKENAFDELING MR 142/72 DECEMBER

J.W. DE BAKKER and L.G.L.Th. MEERTENS

SIMPLE RECURSIVE PROGRAM SCHEMES AND INDUCTIVE ASSERTIONS

2e boerhaavestraat 49 amsterdam

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

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

ABSTRACT

By an unpublished result of Scott, the inductive characterization of the while statement (due to Hoare) is equivalent to its minimal fixed point characterization. In order to obtain a generalization of this result for recursive procedures, a refinement of Floyd's technique of inductive assertions is proposed. The new technique features the use of assertions depending upon the history of the computation. Technically, this is achieved by indexing the assertions with expressions representing the stack of currently active procedures.

The investigation is set in the framework of program schematology. Proofs about simple - i.e., one-variable only - schemes are given by means of Scott's induction rule which is stated and proved somewhat more abstractly and rigorously than before. The main tool is the *regularization theorem* stating, roughly, that for each "context free" program scheme an equivalent (infinite) "regular" scheme can be constructed. The *inductive assertion theorem* then provides the above mentioned generalization.

CONTENTS

1. INTRODUCTION

Our paper reports an investigation of the *foundations* of simple recursive program schemes and their associated inductive assertions.
Simple recursive program schemes were first introduced in Scott and De Bakker [16]. In that paper, the notion of *minimal fixed point* structure of recursive procedures - used synonymously, there and here, for simple recursive program schemes- was developed, and a powerful rule of proof, Scott's *fixed point induction rule*, was derived from it. The variety and multitude of applications of this rule have shown it to be a worthy successor to McCarthy's classical rule of recursion induction [12].
Proofs based on the minimal fixed point characterization were proposed independently by Bekic [4], Morris [13] and Park [15]. In subsequent work De Bakker [1,2], De Bakker and De Roever [3], Hitchcock and Park [6], Manna and Cadiou [8], Manna, Ness and Vuillemin [9], Manna and Vuillemin [11], Milner [14] and others have been concerned with the development of formal systems in which Scott's rule can be embedded, with the completeness of such systems, with application to correctness and termination proofs about programs, with the relationship between the fixed point characterization and various rules of computation, with the implementation of the rule in an interactive program proving system, and with other applications.
Our study of simple recursive program schemes in relation to inductive assertions arose out of a problem inspired by work of Hoare [7]. Incidentally, so did the original paper by Scott and De Bakker. *) The problem is explained in section 2. Roughly, we became interested in the relationship between the inductive characterization of the while statement and its minimal fixed point characterization. The equivalence of these two characterizations was shown by Scott (unpublished). The question then arose how to generalize this result for recursive procedures. The answer to this question is the main achievement of the present paper, beside a number of tech-

*) The reader who takes this reminder as a gentle admonition to the practical program correctness provers, the advocates of structured programming, their company and followers, that there is more to Hoare's axiom system than meets the eye, is right.

niques used in the proof which may have some independent interest. In particular, we develop a strategy for proving properties of programs by means of inductive assertions *depending upon the history of the computation.*
In the course of our investigations we were led to a new development of part of the material contained in the papers Scott and De Bakker [16], and De Bakker and De Roever [3]. The novelty consists mainly in a more abstract and general version of the previous expositions. In particular, we state our results throughout for infinite systems of declarations (this will be needed in section 4), our statement of the union theorem, yielding the construction of the minimal fixed point by means of successive approximations, has a more general form than before, and the induction theorem, justifying Scott's induction rule, is proved without an explicit appeal to continuity. The main new results follow in section 4. There we prove the equivalence between the inductive assertion - and the minimal fixed point characterizations of sytems of recursive procedures. The central tool is a certain indexing technique used first in the proof of the so-called regularization theorem which states that for each recursive program scheme an equivalent (but always infinite) scheme can be constructed which is regular in structure (in the sense that the grammar which is associated with the scheme in a rather natural way is regular). Half of the inductive assertion theorem may be viewed as a justification of a generalization of Floyd's technique [5] of proving global properties of a program from a collection of local properties. This generalization is twofold. Firstly, as minor point, we have that the technique applies to systems of recursive procedures and not to flow charts (cf. the - different - generalization of Manna and Pnueli [10]). Secondly, and more importantly, we construct a system of inductive assertions consisting, in a sense, of the *minimal* set of assumptions about local properties needed to prove the global assertions. The minimality is obtained by introducing assertions which depend upon the stage of the computation: Let A be an elementary component of the program. Usually one requires that, for some pre-(post)condition p(q), if A is entered with input x for which p(x) is true, q(y) is true for output y from A. In our system, however, we use an *indexed* set p_σ, q_σ, σ reflecting the stack of currently active procedures, and require that, for each relevant σ, p_σ, q_σ and A

satisfy the relationship described for p, q and A above. We first prove for this highly structured collection of inductive assertions that Floyd's theorem holds. But, moreover, we can now also prove the converse, i.e., that each system of relations satisfying the collection necessarily coincides with the recursive procedures as declared by the scheme. Thus we obtain the generalization of the result for while statements we set out to prove.

As remarked above, the proofs in section 4 rely heavily on a certain strategy of indexing procedures in various auxiliary systems in such a way that the history of the computation leaves its trace in the index; also, we introduce segments of initial computation, preceding an inner call of a procedure at a given level of recursion depth. The relationship between this notion and the notion of derivative introduced by Hitchcock and Park [6] is settled (without proof) in the Appendix.

For the reader who is not happy with our restriction to simple schemes only, we announce work in progress by W.P. de Roever in which, among other results, section 3 of the present paper is generalized to polyadic relations. Our paper is rather abstract and mathematical in nature. Another unhappy reader, who wants to see what can be done with these techniques in practical programming situations, is referred to the literature mentioned above, e.g. De Bakker [1], De Bakker and De Roever [3], Manna and Vuillemin [11], or Milner [14].

We acknowledge many helpful discussions with P. van Emde Boas. In particular, we are indebted to him for lemma 3.5.

2. ORIGIN OF THE PROBLEM [1)]

Let P be a program. The computation prescribed by P maps input x to output y, with x, y elements of some domain V of state vectors, information structures, internal objects, or whatever one chooses to call them. Articulating the structure of the objects in V is not of our concern here at all; that of P is analysed only in a highly global manner, abstracting from most of the properties of its constituent components. In fact, we study only the essential flow of control structure of P, and investigate it from a mathematical as opposed to an operational or implementation-oriented point of view.
The mapping P is a partially defined (programs may be nonterminating) function from V to V, or, rather, taking non-deterministic programs into account, a *binary relation* over V. We write $(x,y) \in P$, or, more often, xPy. Thus, xPy_1, xPy_2 and $y_1 \neq y_2$ may coexist.
Many correctness assertions on programs can be formulated as: If x satisfies property p, than y satisfies property q, i.e., $\forall x,y[p(x) \wedge xPy \rightarrow q(y)]$.
Concepts of the programming language used for the writing of P can be characterized semantically by correctness assertions. As an example, we consider the *while statement* while p do A, with p a boolean expression, A a program. As short-hand we use p*A. Hoare [7] has proposed what amounts to the following characterizing properties:

(2.1) $$\forall u[\forall x,y[u(x) \wedge p(x) \wedge xAy \rightarrow u(y)] \rightarrow \forall x,y[u(x) \wedge x\ p{*}A\ y \rightarrow u(y)]]$$

(2.2) $$\forall x,y[x\ p{*}A\ y \rightarrow \neg p(y)]$$

In words, (2.1) expresses an induction property: If performing A once (for input with p true) does not change property u, then performing it zero or more times (by p*A) does not change it either. Observe that (2.1) is a

[1)] The considerations of this section are mostly informal in nature. In a more precise form they return in the sequel of the paper.

formula in *second order* predicate logic. (2.2) is clearly valid, since the very termination of the while statement implies that its controlling expression is no longer satisfied.

Formulae such as (2.1), (2.2) can be written more concisely by using a number of abbreviations together establishing a transliteration from predicate - to relational calculus. The predicate $P(x,y)$ is true iff (x,y) is an element of the relation P. Thus, for $\forall x,y\ [xP_1y \rightarrow xP_2y]$ we write $P_1 \subseteq P_2$. The meaning of $P_1 = P_2$, $P_1 \cap P_2$ and $P_1 \cup P_2$ should be clear. $xP_1;P_2y$ is short for $\exists z\ [xP_1z \wedge zP_2y]$; i.e., ";" denotes the operation of relational composition. For the identity (empty) relation we write $E(\Omega)$, i.e., xEy iff $x=y$, and $x\Omega y$ for no $x,y \in \mathcal{V}$. Moreover, with each unary predicate p (possibly partial) we associate two subsets of E, viz. p and $\bar{p}$, such that $p \cup \bar{p} \subseteq E$, $p \cap \bar{p} \subseteq \Omega$, with the following intended correspondence: $p(x)$ holds iff $(x,x) \in p$, $\neg p(x)$ holds iff $(x,x) \in \bar{p}$, and $p(x)$ is undefined iff $(x,x) \in$ $\in E \setminus (p \cup \bar{p})$. Using these abbreviations we can write for (2.1), (2.2):

(2.3) $\forall u\ [\text{If } p;u;A \subseteq A;u \text{ then } u;p{*}A \subseteq p{*}A;u]$

(2.4) $p{*}A \subseteq p{*}A;\bar{p}$.

These two formulae are not yet the whole story about the while statement. Clearly, there is at least one other essential fact to be noted, expressed by

(2.5) $p{*}A = p;A;p{*}A \cup \bar{p}$

or, in perhaps more familiar terms, $p{*}A$ is equivalent with (in fact, may be said to be defined recursively by) if p then begin $A;p{*}A$ end else E, where E is nothing but the "dummy statement". However, (2.5) is not the whole truth either. This will be brought out by consideration of the special case $p{*}E$, where we have taken for the as yet unspecified program A, the dummy statement E. We know that, if p is true of the input, then $p{*}E$ loops infinitely (the relation between input and output being empty in this case), i.e., $p{*}E = \underline{\text{if}}\ p\ \underline{\text{then}}\ \Omega\ \underline{\text{else}}\ E = p;\Omega \cup \bar{p} = \bar{p}$, However, this fact is not contained in the corresponding instance of (2.5). Specifically, (2.5) only expresses that $p{*}A$ is *a* solution of the functional (or, rather, relational) equation $X = p;A;X \cup \bar{p}$, whereas our example emphasizes that we need its *minimal* solution: We have to require

(2.6) $\forall S[\text{If } p;A;S \cup \bar{p} = S, \text{ then } p{*}A \subseteq S]$ [1)]

One is now confronted with the question: What is the relationship between (2.3), (2.4) on the one hand, and (2.6) on the other hand. The answer is provided by the following theorem:

THEOREM 2.1 (Scott). Let R satisfy: $R = p;A;R \cup \bar{p}$. Then the two assertions (2.7a,b) are equivalent with (2.8):

(2.7a) $\forall u[\text{If } p;u;A \subseteq A;u \text{ then } u;R \subseteq R;u]$

(2.7b) $R \subseteq R;\bar{p}$

(2.8) $\forall S[\text{If } p;A;S \cup \bar{p} = S, \text{ then } R \subseteq S]$.

In words, for fixed points R of the while statement characteristic equation, the inductive characterization (2.7) and the minimality characterization (2.8) are equivalent, i.e., imposing either (2.7) or (2.8) upon such R implies that $R = p{*}A$.

PROOF

1. (2.7) $\Rightarrow$ (2.8).
   First we show the following: Let $A^* \overset{\text{df.}}{=} E \cup A \cup A;A \cup \ldots$, and let X be an arbitrary relation over V satisfying: $\forall u[\text{If } u;A \subseteq A;u, \text{ then } u;X \subseteq X;u]$. Then $X \subseteq A^*$. Proof: Choose a fixed $x_0 \in V$. Define $u_0(s) \leftrightarrow \forall t[sA^*t \rightarrow x_0A^*t]$.

[1)] A (generalized) theorem to this effect is proved in section 3.2

It is easily verified, using $A;A^* \subseteq A^*$, that $u_0;A \subseteq A;u_0$. Hence by assumption, $u_0;X \subseteq X;u_0$, or, $\forall x,y[u_0(x) \wedge xXy \rightarrow u_0(y)]$. Assume x_0Xy. Clearly, $u_0(x_0)$ is true. Thus, $u_0(y)$, i.e., $\forall t[yA^*t \rightarrow x_0A^*t]$ holds. Taking $t = y$ we obtain, since $E \subseteq A^*$, the result that $x_0Xy \rightarrow x_0A^*y$. Since x_0 was arbitrary, the proof of $X \subseteq A^*$ is completed. Using this auxiliary result the proof of (2.7) $\Longrightarrow$ (2.8) is easily established as follows: From (2.7a), $\forall u[\text{If } u;(p;A) \subseteq (p;A);u, \text{ then } u;R \subseteq R;u]$. Therefore, $R \subseteq (p;A)^*$, whence, $R;\bar{p} \subseteq (p;A)^*;\bar{p}$, from which, by (2.7b), $R \subseteq (p;A)^*;\bar{p}$ is obtained. Now suppose that $S = p;A;S \cup \bar{p}$. In order to show that then $R \subseteq S$, it is sufficient to show that each of $E;\bar{p}$, $p;A;\bar{p}$, $p;A;p;A;\bar{p},\ldots,$ $(p;A)^i;\bar{p},\ldots$ is included in S. This follows by: $S \supseteq p;A;S \supseteq \ldots \supseteq$ $\supseteq (p;A)^i;S = (p;A)^i;(p;A;S \cup \bar{p}) \supseteq (p;A)^i;\bar{p}$.

2. (2.8) $\Longrightarrow$ (2.7).

By the Knaster-Tarski theorem [18], as mentioned e.g. in de Bakker [1] or Park [15], we have that (2.8) is equivalent with

(2.9) $\quad \forall S[\text{If } p;A;S \cup \bar{p} \subseteq S, \text{ then } R \subseteq S]$.

Let R satisfy (2.8) and, hence, (2.9). Let u be such that $p;u;A \subseteq A;u$. We show that then $u;R \subseteq R;u$, or, equivalently, that $\forall x,y[xRy \rightarrow [u(x) \rightarrow u(y)]]$. Let $xSy \xleftrightarrow{\text{df.}} [u(x) \rightarrow u(y)]$. By (2.9), it will be sufficient to show $p;A;S \cup \bar{p} \subseteq S$. Clearly, $\bar{p} \subseteq S$. Also, in order to show $p;A;S \subseteq S$, we must prove $\forall x,y,z[p(x) \wedge xAy \wedge [u(y) \rightarrow u(z)] \rightarrow$ $\rightarrow [u(x) \rightarrow u(z)]]$. Assume $p(x)$, xAy, $u(y) \rightarrow u(z)$, and $u(x)$. Since $p;u;A \subseteq A;u$, we have $u(y)$. Thus, $u(z)$ follows from the assumption, as desired. This completes the proof of (2.8) $\Longrightarrow$ (2.7a). That of (2.8) $\Longrightarrow$ (2.7b) is left to the reader.

We can now state the origin of the investigation leading to the present paper: We wanted to solve the problem: *Generalize theorem 2.1 for recursive procedures*.

3. SIMPLE RECURSIVE PROGRAM SCHEMES

A simple recursive program scheme is an abstract form of a program containing a system of declarations of recursive procedures. In an ALGOL-like language the structure of such a program might be

```
begin
            procedure P1;<statement 1>;
            procedure P2;<statement 2>;
            ...
            procedure Pn;<statement n>;
            <statement>
end
```

where <statement 1>, ..., <statement n>, and <statement> each may contain occurrences (i.e; "calls") of $P_1, P_2, \ldots, P_n$.
In section 3.1 we first give a precise description of the language in which the abstract statements, i.e., statement *schemes* are written. Informally, the language allows construction from certain elementary statements - either "atomic" actions or procedure calls - by means of composition, denoted by the go-on operator ";", or by means of the union operator "∪". For our use of "∪" compare the previous section, where it was indicated how the conditional statement if p then S_1 else S_2 is represented by $p;S_1 \cup \bar{p};S_2$.
For the moment, we do not yet bring these p's into the formal language. They can wait till section 4.
After the introduction of the formal language, we define how a program scheme written in it can be interpreted as prescribing a *computation*. Starting with an initial interpretation of the atomic actions and the constants (Ω and E) as mappings (relations) over some domain, we construct from this initial interpretation the interpretation of the scheme as a whole, using the notion of *computation sequence*, the definition of which embodies, among others, the "copy rule" for procedures.
Finally, after having prescribed the form of the *assertions* we shall be

interested to make about program schemes, we define the notion of *validity* of assertions. The fundamental theorems about program schemes are then derived in sections 3.2 and 3.3.

3.1 Language and interpretation

The basic components of program schemes are the two classes of symbols introduced in

DEFINITION 3.1 (Basic symbols)

a. The class of *relation symbols* $\mathcal{R} = \mathcal{A} \cup \mathcal{X} \cup \mathcal{C}$, where $\mathcal{A} = \{A_1, A_2, \ldots\}$, $\mathcal{X} = \{X_1, X_2, \ldots\}$, and $\mathcal{C} = \{\Omega, E\}$. Arbitrary elements of $\mathcal{R}$ ($\mathcal{A}$,$\mathcal{X}$) are denoted by $R, R_1, R_2, \ldots$, $(A, A_1, A_2, \ldots, X, X_1, X_2, \ldots)$. The elements of $\mathcal{C}$ are denoted by Ω and E respectively.

b. The class of *procedure symbols* $\mathcal{P} = \{P_1, P_2, \ldots\}$ with arbitrary elements denoted by $P, P_1, P_2, \ldots$.

Remark: The distinction between $\mathcal{A}$ and $\mathcal{X}$ is introduced only for the technical reason of making available a convenient substitution mechanism; as to their interpretation, $\mathcal{A}$ and $\mathcal{X}$ are treated in the same way.
From the classes $\mathcal{R}$ and $\mathcal{P}$ we construct the classes of statement schemes $\mathcal{SS}$, of declaration schemes $\mathcal{DS}$, and of program schemes $\mathcal{PS}$.

DEFINITION 3.2 (Schemes)

a. The class of *statement schemes* $\mathcal{SS}$ (arbitrary elements $S, S_1, \ldots, S', \ldots$):

   1. $\mathcal{R} \cup \mathcal{P} \subseteq \mathcal{SS}$

   2. If $S_1, S_2 \in \mathcal{SS}$, then $(S_1;S_2)$ and $(S_1 \cup S_2) \in \mathcal{SS}$.

b. The class of *declaration schemes* $\mathcal{DS}$ (arbitrary elements D, D_1, ...):
A declaration scheme is a set of pairs $\{P_p, S_p\}_{p\in\pi}$, with π a (not necessarily finite) index set, and, for each $p\in\pi$, $P_p \in \mathcal{P}$, $S_p \in \mathcal{SS}$.

c. The class of *program schemes* $\mathcal{PS}$ (arbitrary elements $T, T_1, \ldots, T', \ldots$):
A program scheme is a pair (D,S) with $D \in \mathcal{DS}$, $S \in \mathcal{SS}$.

A program scheme $T = (D,S) = (\{P_p,S_p\}_{p\in\pi},S)$ will usually be displayed as

$$\begin{array}{l}
\vdots \\
\left.\begin{array}{l} P_p \Longleftarrow S_p \end{array}\right\} p \in \pi \\
\vdots \\
S
\end{array}$$

e.g., for $\pi = \{1,2\}$ we might have

$$\begin{array}{l}
P_1 \Longleftarrow A_1;P_1;A_2;P_2;A_3 \cup A_1;P_2 \cup E \\
P_2 \Longleftarrow A_2;P_1;P_2;A_4 \cup \Omega;P_1 \cup A_5 \\
P_1;A_2;P_2
\end{array}$$

where we have dropped the parentheses of definition 3.2, clause a2. These may be restored by using "associativity" and the convention that ";" has priority over "$\cup$": $S_1;S_2 \cup S_3$ is restored as $((S_1;S_2)\cup S_3)$.
Often, for a program scheme $T = (D,S)$, we shall identify T and S when it is clear from the context which D is meant. $S,S_1,\ldots,T,T_1,\ldots$ will then each range both over *SS* and *PS*.

The language allows us to state certain facts about program schemes in the form of *assertions*:

DEFINITION 3.3 (Assertions)

a. An *atomic formula* is of the form $T_1 \subseteq T_2$, with $T_1,T_2 \in PS$.

b. A *formula* is a set of atomic formulae: $\{T_{1,r} \subseteq T_{2,r}\}_{r\in\rho}$, with ρ a, not necessarily finite, index set.

c. An *assertion* is of the form $\Phi \vdash \psi$, with Φ,ψ formulae.

Examples:

1. $X_2;A_2 \subseteq P_1 \vdash (A_1;X_2 \cup E);A_2 \subseteq P_1$
2. $\{X_{1,r} \subseteq X_{2,r}\}_{r\in\rho} \vdash \{A_1;X_{1,r} \subseteq A_1;X_{2,r}\}_{r\in\rho}$

(No confusion should be caused by the - unavoidable - mixture of object-language and metalanguage in the second assertion).

Remark: $T_1 = T_2$ will be used as abbrevation for $T_1 \subseteq T_2$, $T_2 \subseteq T_1$.

The following notation will be used for *substitution*: For $S,S_1 \in SS$, and $X \in X$, $S_1[S/X]$ denotes the result of substituting S for all occurrences of X in S_1. Also, for π any index set, S, $S_p \in SS$ ($p\in\pi$), and $X_p \in X$ ($p\in\pi$), $S[S_p/X_p]_{p\in\pi}$ denotes the result of simultaneously substituting, for each $p\in\pi$, S_p for all occurrences of X_p in S. The notation is extended in an obvious way to atomic formulae, formulae and assertions. E.g., $(T_1 \subseteq T_2)$ $[S/X]$ is short for $T_1[S/X] \subseteq T_2[S/X]$, and $(\Phi \vdash \psi)[S/X]$ for $\Phi[S/X] \vdash \psi[S/X]$. We emphasize that substitution in a program scheme T = (D,S) takes place only in S and not in D. Without explicit mentioning, use will be made of the chain rule for substitutions: $(S[S_1/X])$ $[S_2/X]$ = $S[S_1[S_2/X]/X]$.

We now relate the program schemes as formal objects to their intended meaning. A program scheme T $\in$ *PS* prescribes a class of computations. By choosing firstly a domain over which the computation is to take place, and secondly the concrete realizations of the relation symbols in *R* over this domain, an *interpretation* - depending upon these choices - is assigned to T. The precise definitions follow in definitions 3.4 to 3.6.

DEFINITION 3.4 (Initial interpretation)

An initial interpretation c_0 is given by its domain *V* (an arbitrary non-empty set) and a mapping (also denoted by c_0) from the elements of *R* to binary relations over *V* satisfying the condition that $c_0(\Omega)$ is the empty relation over *V* and $c_0(E)$ the identity relation.

The extension of an initial interpretation c_0 to an (extended) interpretation c needs the notion of a *computation sequence.*

DEFINITION 3.5 (Computation sequence)
A computation sequence with respect to the declaration scheme $D = \{P_p, S_p\}_{p \in \pi}$ and the initial interpretation c_0 with domain V is a *finite* sequence

$$(3.1) \qquad x_1 \; S_1 \; x_2 \; S_2 \; \ldots \; x_n \; S_n \; x_{n+1}$$

with $n \geq 1$, $x_i \in V$ $(1 \leq i \leq n+1)$, $S_i \in SS$ $(1 \leq i \leq n)$, satisfying the condition:
For each i, $1 \leq i \leq n$, one of the following six cases applies:

a1. $S_i = R$. Then $i = n$, and $(x_i, x_{i+1}) \in c_0(R)$.
 2. $S_i = S' \cup S''$. Then $S_{i+1} = S'$ or $S_{i+1} = S''$, and $x_{i+1} = x_i$.
 3. $S_i = P_p$. Then $S_{i+1} = S_p$, where $(P_p, S_p) \in D$, and $x_{i+1} = x_i$.
b1. $S_i = R;S'$. Then $S_{i+1} = S'$ and $(x_i, x_{i+1}) \in c_0(R)$.
 2. $S_i = (S' \cup S'');S$. Then $S_{i+1} = S';S$ or $S_{i+1} = S'';S$, and $x_{i+1} = x_i$.
 3. $S_i = P_p;S$. Then $S_{i+1} = S_p;S$, where $(P_p, S_p) \in D$, and $x_{i+1} = x_i$.

Example: Let D be

$$P_1 \Longleftarrow A_1;P_1;A_2 \cup A_3;P_2$$
$$P_2 \Longleftarrow A_4;P_2 \cup E.$$

A possible computation sequence with respect to D and a given c_0 is $(S_1 = A_5;P_1)$:

$x_1 \; A_5;P_1 \; x_2 \; P_1 \; x_3 \; A_1;P_1;A_2 \cup A_3;P_2 \; x_4 \; A_1;P_1;A_2 \; x_5 \; P_1;A_2$
$x_6 \; (A_1;P_1;A_2 \cup A_3;P_2);A_2 \; x_7 \; A_3;P_2;A_2 \; x_8 \; P_2;A_2$
$x_9 \; (A_4;P_2 \cup E);A_2 \; x_{10} \; A_4;P_2;A_2 \; x_{11} \; P_2;A_2 \; x_{12} \; (A_4;P_2 \cup E);A_2$
$x_{13} \; E;A_2 \; x_{14} \; A_2 \; x_{15}$
with $(x_1, x_2) \in c_0(A_5)$, $x_2 = x_3$, $x_3 = x_4$, $(x_4, x_5) \in c_0(A_1)$, etc..

Remarks:

1. The definition of computation sequence is an elaboration of a proposal by Scott [17].

2. A computation sequence such as (3.1) may be viewed as follows: Each S_i, $1 \leq i \leq n$, is the program which remains to be executed, at stage i, with current "state" x_i. The execution is completed when the last statement - which is necessarily an element of $\mathcal{R}$ - is performed (clause a1). Clauses a2 and b2 describe a choice between two potential continuations. Clauses a3 and b3 give the *copy rule* for procedures: replace the procedure identifier by its body, and continue with the thus modified program. Clauses b1 to b3 contain the usual meaning of ";" prescribing continuation.

We are now sufficiently prepared to define the *interpretation* of a program scheme.

DEFINITION 3.6 (Interpretation)

Let $T = (D,S)$ be a program scheme and let c_0 be an initial interpretation. Then the interpretation c (which is said to extend c_0) is defined by: For each $x,y \in \mathcal{V}$, $(x,y) \in c(T)$ iff there exists a computation sequence $x_1\ S_1\ x_2\ S_2 \ldots x_n\ S_n\ x_{n+1}$ with respect to D and c_0 such that $x_1 = x$, $x_{n+1} = y$, and $S_1 = S$.

Usually, we are interested in assertions about program schemes which hold for *all* interpretations, i.e., which are *valid*:

DEFINITION 3.7 (Validity)

a. An atomic formula $T_1 \subseteq T_2$ *satisfies* an interpretation c, iff $c(T_1) \subseteq c(T_2)$ holds. If $T_1 \subseteq T_2$ satisfies all c, it is called *valid*.

b. A formula Φ satisfies c (is valid) iff all its elements satisfy c (are valid).

c. An assertion $\Phi \vdash \psi$ such that, for all c, if Φ satisfies c then ψ satisfies c, is called valid.

Remarks:

1. Note the distinction between definition 3.7c and the alternative: $\Phi \vdash \psi$ is called valid iff validity of Φ implies validity of ψ. The

alternative is not adopted.

2. From the definitions it follows that if $\Phi \vdash \psi$ is a valid assertion, for arbitrary S, $(\Phi \vdash \psi)\ [S/X]$ is also valid.

Examples of valid assertions

a. With respect to $D = \{P_1 \Longleftarrow P_1\}$

   $P_1 = \Omega$

b. With respect to $D = \{P_1 \Longleftarrow A_1;P_1 \cup A_2$

   $P_2 \Longleftarrow A_1;P_2 \cup E\}$

   $P_1 = P_2;A_2$, and

   $X_1 \subseteq P_2;A_2 \vdash A_1;X_1 \cup A_2 \subseteq P_2;A_2$

The main result of section 3 is a rule for proving validity of assertions (Scott's induction rule). An important tool in the proof of this rule is the union theorem, dealt with in the next subsection.

3.2 The union theorem

We begin with a simple lemma stating some direct consequences of the definition of interpretation.

LEMMA 3.1 [1)]

a. If $T \in R$, then $c_0(T) = c(T)$
b. $c(T_1;T_2) = c(T_1);c(T_2)$
c. $c(T_1 \cup T_2) = c(T_1) \cup c(T_2)$
d. $c(P_p) = c(S_p)$, for each $p \in \pi$.

PROOF. We prove only part d.

1. $\subseteq$. Assume $(x,y) \in c(P_p)$. Then there is a computation sequence $x_1\ S_1\ x_2\ S_2 \ldots x_n\ S_n\ x_{n+1}$, with $x_1 = x$, $x_{n+1} = y$, and $S_1 = P_p$. By definition 3.4, then $S_2 = S_p$, and $x_2 = x_1$. Therefore, $x_2\ S_2 \ldots x_n\ S_n\ x_{n+1}$ is also a computation sequence; hence, $(x,y) = (x_2,x_{n+1}) \in c(S_p) = c(S'_1)$.

[1)] The lemmas of this and the following subsections always refer to suitably defined statement, declaration, or program schemes. In particular, we always assume given the declaration scheme $D = \{P_p,S_p\}_{p \in \pi}$ such that none of the S_p contains any occurrence of an $X \in X$.

2. $\supseteq$. Assume $(x,y) \in c(S_p)$. Thus, there is a computation sequence $x_1\ S_1\ x_2\ S_2\ \dots\ x_n\ S_n\ x_{n+1}$, with $x_1 = x$, $x_{n+1} = y$, and $S_1 = S_p$. Then the sequence $x'_1\ S'_1\ x'_2\ S'_2\ \dots\ x'_m\ S'_m\ x'_{m+1}$, with $m = n+1$, $S'_1 = P_p$, $S'_i = S_{i-1}$, $i = 2,3,\dots,m$, $x'_1 = x_1$, $x'_i = x_{i-1}$, $i = 2,3,\dots,m+1$, is also a computation sequence, whence $(x,y) \in c(P_p)$ follows.

Remarks.

1. The result of lemma 3.1 d, is not as obvious as it may seem. In fact, it does not necessarily hold in certain treatments of the non-monadic case, as has been pointed out by Manna and Cadiou [8].

2. From the definitions and lemma 3.1, the validity of standard properties of program schemes, such as $\Omega \subseteq T$, $(S_1;S_2);S_3 = S_1;(S_2;S_3)$, $E;T = T$, if $S_1 \subseteq S_2$ then $S;S_1 \subseteq S;S_2$ etc., easily follows. These and similar properties will be used in the sequel without explicit mentioning. We do mention separately the monotonicity property in its two most used forms:

LEMMA 3.2 (Monotonicity)

a. $S_1 \subseteq S_2 \vdash S[S_1/X] \subset S[S_2/X]$

b. $\{S_{1,r} \subseteq S_{2,r}\}_{r\in\rho} \vdash S[S_{1,r}/X_r]_{r\in\rho} \subseteq S[S_{2,r}/X_r]_{r\in\rho}$

but we omit its proof, which proceeds by an inductive argument on the complexity of the statement schemes concerned.

We now come to the more interesting part. First we introduce some auxiliary concepts and notation.

DEFINITION 3.8.

a. A statement scheme S is called *closed* if it contains no occurrences of any $X \in \mathcal{X}$.

b. Let S be a statement scheme. $\tilde{S}$ denotes the result of replacing, in S, all occurrences of a procedure symbol P_p by X_p for each $p \in \pi$.

LEMMA 3.3

a. For closed T, $\tilde{T}[P_p/X_p]_{p\in\pi} = T$

b. For arbitrary T:

$$\{S_p \subseteq P_p\}_{p\in\pi} \vdash \widetilde{T[P_p/X_p]_{p\in\pi}}[S_p/X_p]_{p\in\pi} \subseteq T[S_p/X_p]_{p\in\pi}$$

PROOF. Follows from the definitions, properties of substitution and monotonicity.

Next we need, for each T, two sequences of substitution results $T^{[k]}$ and $T^{(k)}$, $k = 0,1,2,\ldots$.

DEFINITION 3.9

a. $T^{[0]} = T$

$T^{[k+1]} = \tilde{T}\,[S_p^{[k]}/X_p]_{p\in\pi}$, $k = 0,1,2,\ldots$

b. $T^{(0)} = \Omega$

$T^{(k+1)} = \tilde{T}\,[S_p^{(k)}/X_p]_{p\in\pi}$, $k = 0,1,2,\ldots$

We immediately have

LEMMA 3.4

a. $P_p^{(k+1)} = S_p^{(k)}$, $k = 0,1,2,\ldots$

b. $T^{(k+1)} = \widetilde{T^{[k]}}\,[\Omega/X_p]_{p\in\pi}$, $k = 0,1,2,\ldots$

c. $T^{[k+1]} = (T^{[k]})^{[1]}$.

PROOF. a and c are left to the reader.

b. We use induction on k.

(i) $k = 0$. $T^{(1)} = \tilde{T}[S_p^{(0)}/X_p]_{p\in\pi} = \tilde{T}[\Omega/X_p]_{p\in\pi}$

$= \widetilde{T^{[0]}}[\Omega/X_p]_{p\in\pi}$.

(ii) Assume the result for k-1. We have

$$\widetilde{T^{[k]}}[\Omega/X_p]_{p\in\pi} = \widetilde{\tilde{T}[S_p^{[k-1]}/X_p]_{p\in\pi}}\ [\Omega/X_p]_{p\in\pi} =$$

$$\tilde{T}\ [\widetilde{S_p^{[k-1]}}/X_p]_{p\in\pi}\ [\Omega/X_p]_{p\in\pi} =$$

$$\tilde{T}\ [\widetilde{S_p^{[k-1]}}[\Omega/X_p]_{p\in\pi}/X_p]_{p\in\pi} = \text{(ind. hypothesis)}$$

$$\tilde{T}\ [S_p^{(k)}/X_p]_{p\in\pi} = T^{(k+1)}.$$

The next two definitions are preparatory to the three main lemmas of this section, lemmas 3.5 to 3.7. The definitions are of a technical nature and are used only in the proofs of these lemmas.

DEFINITION 3.9 (Executable occurrence)

A procedure symbol P_p is said to occur *executable* in a computation sequence $x_1\ S_1\ x_2\ S_2\ \ldots\ x_n\ S_n\ x_{n+1}$, if, for some i, $1 \leq i \leq n$, $S_i = P_p$ or $S_i = P_p;S$.

DEFINITION 3.10 (to Identify)

Let $x_1\ S_1\ x_2\ S_2\ \ldots\ x_n\ S_n\ x_{n+1}$ be a computation sequence. We say that a procedure symbol P_p occurring in some S contained in S_i, $1 \leq i \leq n$, *directly identifies* the corresponding occurrence of P_p in S contained in S_{i+1}, in each of the following cases

a. $S_i = S \cup S'$ and $S_{i+1} = S$, or $S_i = S' \cup S$, and $S_{i+1} = S$.

b. $S_i = R;S$ and $S_{i+1} = S$.

c. $S_i = (S' \cup S'');S$ and $S_{i+1} = S';S$ or $S'';S$, or $S_i = (S \cup S');S''$ and $S_{i+1} = S;S''$, or $S_i = (S' \cup S);S''$ and $S_{i+1} = S;S''$.

d. $S_i = P_q;S$ and $S_{i+1} = S_q;S$, for some $q \in \pi$.

The relationship *to identify* is defined as the reflexive and transitive closure of the relationship *to identify directly*.

LEMMA 3.5 (Van Emde Boas)

Let

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

be a computation sequence with $\delta > 0$ executable occurrences of a procedure symbol. Moreover, we assume that S_1 (and, therefore, each S_i, $i \geq 2$) is a closed statement scheme. Then there exists a computation sequence $x'_1\ S'_1\ x'_2\ S'_2\ \ldots\ x'_m\ S'_m\ x'_{m+1}$, such that $x'_1 = x_1$, $x'_{m+1} = x_{n+1}$, $S'_1 = S_1^{[1]}$, and, moreover, such that for the number δ' of executable occurrences of a procedure symbol in this sequence we have $\delta' \leq \delta-1$.

PROOF. We introduce the following transformation on the computation sequence (3.2):

Step 1. Consider, for each $p \in \pi$, all occurrences of the procedure symbol P_p in (3.2) which are identified by an occurrence of P_p in S_1.

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

Step 3. Replace all other considered occurrences of P_p by S_p, for each $p \in \pi$.

Step 4. Replace, for each $p \in \pi$, all combinations $\ldots x_j\ P^*_p;S\ x_{j+1}\ S_p;S\ x_{j+2} \ldots$ or $\ldots\ x_j\ P^*_p\ x_{j+1}\ S_p\ x_{j+2}\ \ldots$, where P^*_p is an occurrence of P_p, marked as a result of Step 2, by $\ldots\ x_j S_p;S\ x_{j+2}\ \ldots$, or by $\ldots\ x_j\ S_p\ x_{j+2}\ \ldots$, respectively.

It can be verified that the result of applying this transformation to (3.2) is again a computation sequence which has at least one executable occurrence of some P_p less than (3.2). In fact, at least the left-most executable occurrence of this P_p has been deleted. Moreover, it is clear that for the resulting sequence we have, by step 3 or 4, that $S'_1 = \tilde{S}_1[S_p/X_p]_{p\in\pi} = S_1^{[1]}$.

LEMMA 3.6

Let $x_1\ S_1\ x_2\ S_2\ \ldots\ x_n\ S_n\ x_{n+1}$ be a computation sequence with closed S_i, $1 \leq i \leq n$, and without any executable occurrence of a procedure symbol. Then, for arbitrary $R_p \in \mathcal{R}$, $p \in \pi$, we have that

$$x_1\ \tilde{S}_1[R_p/X_p]_{p\in\pi}\ x_2\ \tilde{S}_2[R_p/X_p]_{p\in\pi}\ \ldots\ x_n\ \tilde{S}_n[R_p/X_p]_{p\in\pi}\ x_{n+1}$$

is also a computation sequence.

PROOF. Since none of the P_p is executable, each of its occurrences may be replaced by an arbitrary R_p without changing the computation.

LEMMA 3.7

Let T be a closed statement scheme, and let $(x,y) \in c(T)$. Then there exists $k > 0$ such that $(x,y) \in c(T^{(k)})$.

PROOF. By assumption, there is a computation sequence $x_1\ S_1\ x_2\ S_2 \ldots$ $\ldots\ x_n\ S_n\ x_{n+1}$, with $x_1 = x$, $x_{n+1} = y$, and $S_1 = T$. Since $S_1 = T$ is closed, each S_i is closed. Repeatedly applying lemmas 3.5 and 3.4c yields, for some $k > 0$, a computation sequence $x'_1\ S_1^{[k]}\ x' \ldots x'_m\ S'_m\ x'_{m+1}$, such that $x'_1 = x_1$, $x'_{m+1} = y$, and such that this computation sequence does not contain any executable occurrence of a procedure symbol. Then, by lemma 3.6, we have that

$$x'_1\ \widetilde{S_1^{[k]}}[\Omega/X_p]_{p\in\pi}\ x'_2\ \ldots\ x'_m\ \tilde{S}'_m[\Omega/X_p]_{p\in\pi}\ x'_{m+1}$$

is also a computation sequence. By lemma 3.4, part b, $\widetilde{S_1^{[k]}}[\Omega/X_p]_{p\in\pi} = S_1^{(k+1)}$. Thus, we have shown that $(x,y) \in c(S_1^{(k+1)})$.

LEMMA 3.7 provides the main result for the proof of

THEOREM 3.1 (Union theorem)

Let T be a closed statement scheme. Then, for all c,

$$c(T) = \bigcup_{k=0}^{\infty} c(T^{(k)}).$$

PROOF.

a. $\subseteq$. This follows directly from lemma 3.7.

b. $\supseteq$. First we show that, for each $p \in \pi$, and each k, $P_p^{(k)} \subseteq P_p$. We use induction on k.

(i) $k = 0$. Clear.

(ii) Assume the result for k. Then: $P_p^{(k+1)} =$ (lemma 3.4)

$$S_p^{(k)} = \tilde{S}_p[S_p^{(k-1)}/X_p]_{p\in\pi} = \tilde{S}_p[P_p^{(k)}/X_p]_{p\in\pi} \subseteq$$

(ind. hypothesis) $\tilde{S}_p[P_p/X_p]_{p\in\pi} = S_p =$ (lemma 3.1) P_p.

Next, we show that $T^{(k)} \subseteq T$: $T^{(k)} = \tilde{T}[S_p^{(k)}/X_p]_{p\in\pi} = \tilde{T}[P_p^{(k+1)}/X_p] \subseteq$ $\subseteq \tilde{T}[P_p/X_p] =$ (lemma 3.3) T. Thus, $\bigcup_{k=0}^{\infty} T^{(k)} \subseteq T$ follows, whence the proof of part b.

Remark: In the sequel we shall abbreviate the statement "For all c, $c(T) = \bigcup_{k=0}^{\infty} c(T^{(k)})$" to: $T = \bigcup_{k=0}^{\infty} T^{(k)}$.

As a corollary to theorem 3.1, we immediately obtain the minimal fixed point property of procedures:

COROLLARY 3.1

$$\{\tilde{S}_p[S'_p/X_p]_{p\in\pi} \subseteq S'_p\}_{p\in\pi} \vdash \{P_p \subseteq S'_p\}_{p\in\pi}.$$

PROOF. We use $P_p = \bigcup_{k=0}^{\infty} P_p^{(k)}$ and induction on k.

(i) $P_p^{(0)} \subseteq S'_p$ is clear.

(ii) Assume the result for k. Then, for each $p \in \pi$, $P_p^{(k+1)} = S_p^{(k)} =$

$$\tilde{S}_p[S_p^{(k+1)}/X_p]_{p\in\pi} \subseteq \text{(ind. hypothesis) } \tilde{S}_p[S'_p/X_p]_{p\in\pi} \subseteq S'_p.$$

Finally, we are now in a position to prove the induction theorem, the importance of which justifies devoting a separate section to it:

3.2. <u>The induction theorem</u>

THEOREM 3.2 (Scott's induction theorem)

Let Φ be a closed formula. Then:

If

$$\Phi \vdash \psi[\Omega/X_p]_{p\in\pi}$$

and

$$\Phi,\Psi \vdash \Psi[S_p/X_p]_{p\in\pi}$$

are valid, then

$$\Phi \vdash \Psi[P_p/X_p]_{p\in\pi}$$

is valid.

PROOF. It is sufficient to show the following:
If (*) $(T_1\subseteq T_2)[\Omega/X_p]_{p\in\pi}$, and (**) $T_1 \subseteq T_2 \vdash (T_1\subseteq T_2)[\tilde{S}_p/X_p]_{p\in\pi}$ are valid, then $(T_1\subseteq T_2)[P_p/X_p]_{p\in\pi}$ is valid. Observe that the T_1, T_2 may contain occurrences of the P_p; in other words, we do not necessarily have that $\widetilde{T_i[P_p/X_p]}_{p\in\pi} = T_i$, i = 1,2,... . The proof proceeds in four steps:

a. We show that $T_1[S_p^{(k)}/X_p]_{p\in\pi} \subseteq T_2[S_p^{(k)}/X_p]_{p\in\pi}$, k = 0,1,2,..., by induction on k. The case k = 0 follows from (*). Next, assume as induction hypothesis that $T_1[S_p^{(k)}/X_p]_{p\in\pi} \subseteq T_2[S_p^{(k)}/X_p]_{p\in\pi}$ holds. By (**) we have that $\{T_1 \subseteq T_2 \vdash (T_1\subseteq T_2)[S_p/X_p]_{p\in\pi}\}[S_p^{(k)}/X_p]_{p\in\pi}$. From this, $(T_1\subseteq T_2)[S_p^{(k)}/X_p]_{p\in\pi} \vdash (T_1\subseteq T_2)[S_p^{(k+1)}/X_p]_{p\in\pi}$ follows. Combination with the induction hypothesis yields that $(T_1\subseteq T_2)[S_p^{(k+1)}/X_p]_{p\in\pi}$ holds.

b. For k = 0,1,2,..., and any T, we have $(T[P_p/X_p]_{p\in\pi})^{(k+1)} \subseteq T[S_p^{(k)}/X_p]_{p\in\pi}$, since $(T[P_p/X_p]_{p\in\pi})^{k+1} = \widetilde{T[P_p/X_p]_{p\in\pi}}[S_p^{(k)}/X_p]_{p\in\pi} \subseteq T[S_p^{(k)}/X_p]_{p\in\pi}$, by lemma 3.4.

c. By b, $\bigcup_{k=0}^{\infty} (T[P_p/X_p]_{p\in\pi})^{(k+1)} \subseteq \bigcup_{k=0}^{\infty} T[S_p^{(k)}/X_p]_{p\in\pi} \subseteq T[P_p/X_p]_{p\in\pi}$. Also, by theorem 3.1, which applies since $T[P_p/X_p]_{p\in\pi}$ is closed, we have that $\bigcup_{k=0}^{\infty} (T[P_p/X_p]_{p\in\pi})^{(k+1)} = T[P_p/X_p]_{p\in\pi}$. Thus, we obtain that $\bigcup_{k=0}^{\infty} T[S_p^{(k)}/X_p]_{p\in\pi} = T[P_p/X_p]_{p\in\pi}$.

d. Combination of parts a and c completes the proof of the induction theorem.

Example: Let $\pi = \{1,2\}$ and $D = \{P_1 \Longleftarrow A_1;P_1 \cup A_2,$

$$P_2 \Longleftarrow A_1;P_2 \cup E\}.$$

We show that $P_1 = P_2;A_2$ (this standard example was used first in [16]).

1. $\subseteq$. Take for Φ the empty list and for ψ: $X_1 \subseteq P_2;A_2$. Then, for this ψ, $\psi[\Omega/X_p]_{p\in\{1,2\}}$ is the assertion $\Omega \subseteq P_2;A_2$, which is clearly valid. Next, we have as instance of $\psi \vdash \psi[\tilde{S}_p/X_p]_{p\ \pi}$:

$$X_1 \subseteq P_2;A_2 \vdash A_1;X_1 \cup A_2 \subseteq P_2;A_2.$$

Since, by lemma 3.1.d, $P_2 = A_1;P_2 \cup E$, we must prove

$$X_1 \subseteq P_2;A_2 \vdash A_1;X_1 \cup A_2 \subseteq (A_1;P_2 \cup E);A_2 = A_1;P_2;A_2 \cup A_2$$

which is valid by monotonicity. We conclude that $\psi[P_p/X_p]_{p\in\{1,2\}}$, i.e., $P_1 \subseteq P_2;A_2$, holds.

2. $\supseteq$. Take Φ again empty and for ψ: $X_2;A_2 \subseteq P_1$. Validity of $\psi[\Omega/X_p]_{p\in\{1,2\}}$ is clear. Also, $\psi \vdash \psi[\tilde{S}_p/X_p]_{p\in\{1,2\}}$ takes the form

$$X_2;A_2 \subseteq P_1 \vdash (A_1;X_2 \cup E);A_2 \subseteq P_1 \ (= A_1;P_1 \cup A_2)$$

and the desired result follows again by monotonicity, implying, by theorem 3.2, the validity of $\psi[P_p/X_p]_{p\in\{1,2\}}$, i.e., of $P_2;A_2 \subseteq P_1$.

A large number of examples, in varying degrees of difficulty, of applying the rule, is contained in the papers mentioned at the end of the introduction. Section 4 will provide another - more advanced - application.

4. INDUCTIVE ASSERTIONS

In this section we introduce the notion of a system of inductive assertions associated with a simple recursive program scheme, and we prove the main theorem about them which states the equivalence of characterizing recursive procedures in terms of inductive assertions, and in terms of the minimality of fixed points.

Our terminology is derived from the "inductive assertion method" of Floyd [5], which may be viewed as a technique for deriving global properties of a program from local properties of its components. The form in which this method is presented here is more abstract and general than the usual one. Observe that our description of it in the framework of recursive program schemes has the flow chart definition as a special case (each flow chart can be described by a (regular, see section 4.2) system of recursive procedures). Note also that the usual requirement of having at least one assertion "breaking each loop" for the flow chart case has no counterpart here, since it is dealt with automatically if a system of recursive procedures is associated in the usual way with a flow chart.

One half of the main theorem (theorem 4.2, part 1) is a generalization of theorem 6.1 from De Bakker and De Roever [3].

Our formal treatment of Floyd's method needs an extension of our formal language in order to deal with the entrance - and exit conditions of the program and its components.

Therefore, we extend the formal language by adding to R a special class of relation symbols, the class $A_p = \{p_1, p_2, \dots\}$ of predicate symbols, arbitrary elements of which are denoted by $p, p_1, \dots, q, q_1, \dots$. This extension of R needs an extension of the definition of initial interpretation (definition 3.4): We require that, for each $p \in A_p$, $c_0(p) \subseteq c_0(E)$; i.e., each p is interpreted as a subset of the identity relation. In this way we can find, for each inductive assertion formulated as a sentence in predicate calculus: $\forall x,y[p(x) \wedge xAy \rightarrow q(y)]$ an equivalent formula in our language: $p;A \subseteq A;q$, with the property that, for each model in which this sentence is true, we can find an initial interpretation c_0 with extension c such $p;A \subseteq A;q$ satisfies c, and vice versa.

With section 4.1 we hope to provide the reader with some feeling for the problem of proving the second half of our main theorem (theorem 4.2, part 2).

4.1. Attempts that failed

In section 2, we considered the while statement $p*A = p;A;p*A \cup \bar{p}$. In terms of program schemes, the characterizing theorem for while statements (theorem 2.1) can be reformulated as: Let P be declared by: $P \Leftarrow A_1;P \cup A_2$. Then for each T, the following assertion

$$\left.\begin{array}{l} p;A_1 \subseteq A_1;p \\ p;A_2 \subseteq A_2;q \end{array}\right| \vdash p;T \subseteq T;q$$

is equivalent with $T \subseteq P$. Now for its generalization. Let us consider P_1 declared by $P_1 \Leftarrow A_1;P_1;A_2 \cup A_3$. One might, as first attempt, try to prove the equivalence of $T \subseteq P_1$ and

$$\left.\begin{array}{l} p;A_1 \subseteq A_1;p \\ q;A_2 \subseteq A_2;q \\ p;A_3 \subseteq A_3;q \end{array}\right| \vdash p;T \subseteq T;q$$

but this fails. E.g., $T = A_1;A_2;A_3;A_2$ satisfies the inductive assertions requirement, but it is not true that $T \subseteq P_1$. As next trial we use an infinity of p_i,q_i, $i = 0,1,2,\ldots$, each i reflecting the current recursion depth:

$$\left.\left\{\begin{array}{l} p_i;A_1 \subseteq A_1;p_{i+1} \\ q_{i+1};A_2 \subseteq A_2;q_i \\ p_i;A_3 \subseteq A_3;q_i \end{array}\right\}_{i=0,1,2,\ldots}\right| \vdash \{p_i;T \subseteq T;q_i\}_{i=0,1,2,\ldots}$$

and, indeed, $T \subseteq P_1$ is now valid. How to generalize this once more? Consider $P_2 \Leftarrow A_1;P_2;A_2;P_2;A_3 \cup A_4$. Directly taking over the $\{p_i,q_i\}_{i=0,1,\ldots}$ approach is easily seen to fail. One soon realizes that one has to distinguish the two occurrences of P_2 at the right hand side, and one might try to use two systems $\{p_i,q_i\}_{i=0,1,\ldots}$, and $\{r_i,s_i\}_{i=0,1,\ldots}$, with assertion

$$\left\{\begin{array}{l} p_i;A_1 \subseteq A_1;p_{i+1} \\ r_i;A_1 \subseteq A_1;p_{i+1} \\ q_{i+1};A_2 \subseteq A_2;r_{i+1} \\ s_{i+1};A_3 \subseteq A_3;q_i \\ s_{i+1};A_3 \subseteq A_3;s_i \\ p_i;A_4 \subseteq A_4;q_i \\ r_i;A_4 \subseteq A_4;s_i \end{array}\right\}_{i=0,1,\ldots} \vdash \left\{\begin{array}{l} p_i;T \subseteq T;q_i \\ r_i;T \subseteq T;s_i \end{array}\right\}_{i=0,1,\ldots} .$$

This does not work either. Counterexample: $T = A_1;A_4;A_2;A_4;A_3;A_3;A_3;A_2;A_1;A_4$. So far for the attempts that failed. The reader may have developed some understanding for the complexity of the remaining sections, in particular for the need to refine the indexing strategy for the predicates in order to keep a closer eye on the history of the computation.
The successful attempt begins with the development of the important auxiliary theorem of the next subsection.

4.2. The regularization theorem

Consider the declaration scheme $D = \{P_p,S_p\}_{p\in\pi}$, with each S_p a statement scheme over $\{P_p\}_{p\in\pi} \cup \mathcal{R}$. There is a natural correspondence between D and a (infinite, if π is infinite) context free grammar G, established as follows: $\mathcal{R}$ is the class of terminal symbols of G, $\{P_p\}_{p\in\pi}$ is the class of nonterminals, D' is its set of production rules, where D' is obtained from D by rewriting $S_1;(S_2 \cup S_3)$ as $S_1;S_2 \cup S_1,S_3$, by dropping everywhere the ";"s, and by replacing, e.g., $P_p \Longleftarrow S_1 \cup S_2$ by the two production rules $P_p \rightarrow S_1$, $P_p \rightarrow S_2$, etc.. As designated nonterminal of G any P_p may be selected. Clearly, the typology of grammars carries over to declaration schemes. In particular, this gives us the notion of a *regular* scheme: D is regular, iff its corresponding grammar G is regular. E.g., with reference to subsection

4.1, the scheme for P is regular, but those for P_1 and P_2 are not. The theorem of this subsection tells us, roughly speaking, that for each (finite or infinite) context free declaration scheme an equivalent (but always infinite) regular declaration scheme can be constructed. This theorem will be the main tool in our proof of the inductive assertion theorem below.

THEOREM 4.1 (The regularization theorem)

Let π be an index set, and let $D_\pi = \{P_p, S_p\}_{p\in\pi}$ be a closed declaration scheme, with each S_p a statement scheme over $\{P_p\}_{p\in\pi} \cup R$. Then there is an index set ρ and a closed declaration scheme $D_\rho = \{P_r, S_r\}_{r\in\rho}$, each S_r a statement scheme over $\{P_r\}_{r\in\rho} \cup R$, such that

a. D_ρ is regular.

b. There is a mapping λ from π into ρ such that $P_{\lambda(p)} = P_p$, for each $p \in \pi$.

(The last equivalence should be understood as stating equivalence under all interpretations based upon computation sequences with respect to the declaration scheme $D = D_\pi \cup D_\rho$.)

PROOF. By, if necessary, repeatedly applying $S;(S' \cup S") = S;S' \cup S;S"$, we may assume that, for each $p \in \pi$, S_p has the form

$$(4.1)\qquad S_p = S_{p,1} \cup S_{p,2} \cup \ldots \cup S_{p,M_p},$$

where M_p is some integer ≥ 1, and where, for each $p \in \pi$, $1 \leq j \leq M_p$, $S_{p,j}$ has the form [1] (raising subscripts for typographical reasons):

$$(4.2)\qquad S(p,j) = R(p,j,0);P(p,j,1);R(p,j,1);\ldots;P(p,j,K_{p,j});R(p,j,K_{p,j})$$

with $K_{p,j}$ some integer ≥ 0, with $R(p,j,k) \in R$, $0 \leq k \leq K_{p,j}$, and $P(p,j,k) \in \{P_p\}_{p\in\pi}$, $1 \leq k \leq K_{p,j}$.

[1] Observe that it may be necessary to insert some E's or auxiliary P's declared as E, in the originally given S_p, in order to obtain this form

Let us put

$$\Sigma_0 = \{(p,j,k) \mid p \in \pi,\ 1 \leq j \leq M_p,\ 1 \leq k \leq K_{p,j}\}$$

and let us define the function h: $\Sigma_0 \to \pi$ by $h(p,j,k) = q$ iff $P(p,j,k) = q$. Observe that each occurrence of a procedure symbol P_q in some S_p is uniquely identified by the index triple (p,j,k).

Example: Let D be: $\{P_1 \Longleftarrow A_1;P_1;A_2;P_2;A_3 \cup A_4;P_2;A_5,$
$P_2 \Longleftarrow A_6;P_1;A_7 \cup A_8\}$

Then $\Sigma_0 = \{(1,1,1),(1,1,2),(1,2,1),(2,1,1)\}$, and $h(1,1,1) = 1$, $h(1,1,2) = 2$, $h(1,2,1) = 2$, and $h(2,1,1) = 1$. Let Σ_0^* be the set of all finite sequences of elements of Σ_0, including the empty word ε. We define the language Σ, consisting of words in Σ_0^*, by means of a context free grammar with productions

$$\sigma \to \varepsilon$$

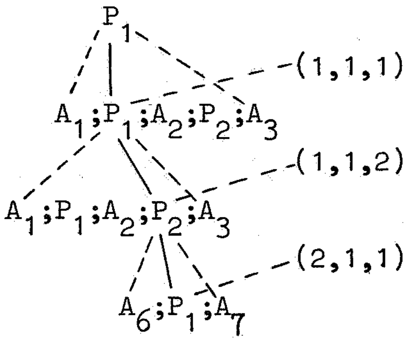
$$\{\sigma \to \sigma_p\}_{p\in\pi}$$

$$\left\{\begin{matrix} \sigma_p \to (p,j,k) \\ \sigma_p \to (p,j,k)\ \sigma_{h(p,j,k)} \end{matrix}\right\}_{(p,j,k)\in\Sigma_0}$$

Σ is the collection of all words in Σ_0^* produced by σ. σ will also be used to denote an arbitrary element of Σ.

Example: For the D just mentioned, possible σ 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.

Observe that each $\sigma \in \Sigma$, produced with an application of the rule $\sigma \to \sigma_p$ as first step, may be viewed as defining a path in the tree of incarnations of the procedures of the system with P_p as root, or, alternatively, σ represents the stack of currently active procedures, each triple in σ representing one procedure call. Compare the following figure (with respect to D again)



The sequence $\sigma = (1,1,1)(1,1,2)(2,1,1)$ represents the calling structure indicated by the drawn lines, where the first component of the first triple in σ - here 1 - is the index of the root of the tree.

Σ is used in the construction of the index set ρ we are looking for in the following manner: ρ is defined as:

$$\rho = \pi \times \Sigma \times \{1,2\}$$

i.e., each P_r, $r \in \rho$, is of one of the two forms $P_{(p,\sigma,1)}$, or $P_{(p,\sigma,2)}$, with $p \in \pi$, $\sigma \in \Sigma$, and $1,2 \in \{1,2\}$.

Moreover, we define, for each $p \in \pi$, $\lambda(p)$ as: $\lambda(p) = (p,\varepsilon,2) \in \rho$. For easier readibility, we use the notation P^p_σ for $P_{(p,\sigma,1)}$, and Q^p_σ for $P_{(p,\sigma,2)}$. Thus, in this notation

$$P_{\lambda(p)} = P_{(p,\varepsilon,2)} = Q^p_\varepsilon.$$

We now have to define, for each $r \in \rho$, the statement scheme S_r, and to prove that the system $\{P_r,S_r\}_{r\in\rho}$ has the desired properties, in particular, that $P_p = Q^p_\varepsilon$.

The definition of the S_r, for $r = (p,\sigma,1)$, is given inductively on the length of the σ:

$$(4.3) \qquad \begin{cases} P^p_\varepsilon \Longleftarrow E \\ P^{h(p,j,1)}_{\sigma(p,j,1)} \Longleftarrow P^p_\sigma;R(p,j,0) \\ P^{h(p,j,k+1)}_{\sigma(p,j,k+1)} \Longleftarrow Q^{h(p,j,k)}_{\sigma(p,j,k)};R(p,j,k), \qquad \text{for } 1 \leq k \leq K_{p,j}-1 \end{cases}$$

and for $r = (p,\sigma,2)$ by

$$(4.4) \qquad Q^p_\sigma \Longleftarrow \bigcup_{j=1}^{M_p} \begin{cases} P^p_\sigma;R(p,j,0) & \text{if } K_{p,j} = 0 \\ \\ Q_{\sigma(p,j,K_{p,j})}^{h(p,j,K_{p,j})};R(p,j,K_{p,j}) & \text{if } K_{p,j} > 0 \end{cases}$$

Example: Let $\pi = \{1,2\}$, and let P be declared by $P \Longleftarrow A_1;P;A_2;P;A_3 \cup A_4$. We have, omitting complications in the indices which are unnecessary for this simple scheme, and taking $\Sigma = \{0,1\}^*$, as regular scheme:

$$P_\varepsilon \Longleftarrow E$$

$$P_{\sigma 0} \Longleftarrow P_\sigma;A_1$$

$$P_{\sigma 1} \Longleftarrow Q_{\sigma 0};A_2$$

$$Q_\sigma \Longleftarrow Q_{\sigma 1};A_3 \cup P_\sigma;A_4.$$

From these definitions we have:

$$\begin{aligned} Q_\varepsilon &= Q_1;A_3 \cup P_\varepsilon;A_4 = (Q_{11};A_3 \cup P_1;A_4);A_3 \cup A_4 \\ &= Q_{11};A_3;A_3 \cup Q_0;A_2;A_4;A_3 \cup A_4 \\ &= \ldots \cup (Q_{01};A_3 \cup P_0;A_4);A_2;A_4;A_3 \cup A_4 \\ &= \ldots \cup \ldots \cup P_0;A_4;A_2;A_4;A_3 \cup A_4 \\ &= \ldots \cup \ldots \cup P_\varepsilon;A_1;A_4;A_2;A_4;A_3 \cup A_4 \\ &= \ldots \cup \ldots \cup A_1;A_4;A_2;A_4;A_3 \cup A_4. \end{aligned}$$

This suggests that $Q_\varepsilon = P$, as will indeed follow by the theorem we are in the process of proving.

Remarks:

1. Observe the distinction between the notation $h(p,j,k)$ denoting the result of applying the function h to $(p,j,k) \in \Sigma_0$, yielding an element of π, and $\sigma(p,j,k)$ denoting the result of concatenating the elements $\sigma \in \Sigma$ and $(p,j,k) \in \Sigma_0$.

2. For each $p \in \pi$, $\sigma \in \Sigma$, P^p_σ has the following intuitive meaning. Let, for some $s \geq 0$, $\sigma = (p_0,j_0,k_0) \ldots (p_s,j_s,k_s)$, with $p = h(p_s,j_s,k_s)$. As we saw above, σ keeps track of a specific path through the tree of incarnation with P_{p_0} as root, leading to the inner call of P_p. Then the computation prescribed by P^p_σ is precisely the computation starting with the outermost call of P_{p_0}, and up to, but not including, this inner call.

   Example: Referring to the figure on page 28 we have

   $$P^1_{(1,1,1)(1,1,2)(2,1,1)} = A_1;A_1;P_1;A_2;A_6.$$

3. (As we shall show below) $Q^p_\sigma = P^p_\sigma;P_p$, so with P^p_σ equivalent to the computation preceding the inner call of P_p with history σ, Q^p_σ is equivalent to this computation but *including* the innercall of P_p.

Once we have shown $Q^p_\sigma = P^p_\sigma\ ;P_p$, we will have obtained our goal, since, for the special case $\sigma = \varepsilon$, $Q^p_\varepsilon = P^p_\varepsilon;P_p$; hence, by the definition of P^p_ε as E, we obtain

$$P_p = Q^p_\varepsilon = P_{(p,\varepsilon,2)} = P_{\lambda(p)}.$$

Moreover, from (4.3) and (4.4) it is immediate that D_ρ is regular.
The next step is the definition of another system of procedures over the same index set ρ: $\bar{D}_\rho = \{\bar{P}_r,\bar{S}_r\}_{r\epsilon\rho}$, as follows:

For $r = (p,\sigma,1)$, the $\bar{S}_r$ are (apart from the procedure symbols) the same as S_r:

$$(4.5)\quad \begin{cases} \bar{P}^p_\varepsilon \Leftarrow E \\ \bar{P}^{h(p,j,1)}_{\sigma(p,j,1)} \Leftarrow \bar{P}^p_\sigma;R(p,j,0) \\ \bar{P}^{h(p,j,k+1)}_{\sigma(p,j,k+1)} \Leftarrow \bar{Q}^{h(p,j,k)}_{\sigma(p,j,k)};R(p,j,k), \qquad 1 \le k \le K_{p,j}-1 \end{cases}$$

but for $r = (p,\sigma,2)$ we have different definitions:

$$(4.6)\qquad \bar{Q}^p_\sigma \Leftarrow \bar{P}^p_\sigma;P_p.$$

We shall show that, for each $r \in \rho$, $\bar{P}_r = P_r$, i.e., for each $p \in \pi$, $\sigma \in \Sigma$,

$$P^p_\sigma = \bar{P}^p_\sigma,\quad Q^p_\sigma = \bar{Q}^p_\sigma.$$

Combining this with (4.6) will yield $Q^p = P^p;P_p$, as desired.

Part 1. $\{P_r \subseteq \bar{P}_r\}_{r\epsilon\rho}$.

By the corollary of theorem 3.1, it is sufficient to show that $\{\bar{P}_r\}_{r\epsilon\rho}$ satisfies

$$(4.7)\qquad \{\tilde{S}_r[\bar{P}_r/X_r]_{r\epsilon\rho} \subseteq \bar{P}_r\}_{r\epsilon\rho}.$$

a. If $r = (p,\sigma,1)$, this is immediate, since, by definitions (4.3) and (4.5), $\tilde{S}_r = \tilde{\tilde{S}}_r$; hence

$$\{\tilde{S}_r[\bar{P}_r/X_r]_{r\epsilon\rho} = \tilde{\tilde{S}}_r[\bar{P}_r/X_r]_{r\epsilon\rho} = \bar{P}_r\}_{r\epsilon\rho},$$

where the last equivalence follows by the fixed point property (lemma 3.1c).

b. Let $r = (p,\sigma,2)$. For each j, $1 \le j \le M_p$, we distinguish two cases:

b1. $K_{p,j} = 0$. Then, by (4.1), (4.2), $R(p,j,0) = S(p,j) \subseteq S_p = P_p$. Hence, $\bar{P}^p_\sigma;R(p,j,0) \subseteq \bar{P}^p_\sigma;P_p = \bar{Q}^p_\sigma$, using (4.6).

b2. $K_{p,j} > 0$. Then,

$$\begin{aligned}
\bar{Q}^p_\sigma = \bar{P}^p_\sigma;P_p &\overset{(4.2)}{\supseteq} \bar{P}^p_\sigma;R(p,j,0);P(p,j,1);...;R(p,j,K_{p,j}) \\
&\overset{(4.5)}{\supseteq} \bar{P}^{h(p,j,1)}_{\sigma(p,j,1)};P(p,j,1);,,,;R(p,j,K_{p,j}) \\
&\overset{(4.6)}{\supseteq} \bar{Q}^{h(p,j,1)}_{\sigma(p,j,1)};R(p,j,1);,,,;R(p,j,K_{p,j}) \\
&\overset{(4.5)}{\supseteq} \bar{P}^{h(p,j,2)}_{\sigma(p,j,2)};P(p,j,2);...;R(p,j,K_{p,j}) \\
&\supseteq \;... \\
&\supseteq \bar{P}^{h(p,j,K_{p,j})}_{\sigma(p,j,K_{p,j})};P(p,j,K_{p,j});R(p,j,K_{p,j}) \\
&\overset{(4.6)}{\supseteq} \bar{Q}^{h(p,j,K_{p,j})}_{\sigma(p,j,K_{p,j})};R(p,j,K_{p,j})
\end{aligned}$$

From b1 and b2 we see that (4.7) is indeed satisfied.

Part 2. $\{\bar{P}_r \subseteq P_r\}_{r\in\rho}$.

Again, by the corollary of theorem 3.1, it is sufficient to show that the $\{P_r\}_{r\in\rho}$ satisfy

(4.8) $\{\tilde{\bar{S}}_r[P_r/X_r]_{r\in\rho} \subseteq P_r\}_{r\in\rho}$.

As in part 1, this is clear for the P^p_σ. In order to show this for the Q^p_σ, we apply the induction theorem 3.2 in the following form: Let Φ be empty, and let ψ be:

$$\{P^p_\sigma;X_p \subseteq Q^p_\sigma\}_{p\in\pi,\sigma\in\Sigma}.$$

That $\psi[\Omega/X_p]_{p\in\pi}$, i.e., $\{P^p_\sigma;\Omega\subseteq Q^p_\sigma\}_{p\in\pi,\sigma\in\Sigma}$, is valid, is clear. Let us put $X(p,j,k) = X_q$, iff $h(p,j,k) = q$. In order to verify the second assumption of the induction theorem in this case, we have to show: If (*) $\{P^p_\sigma;X_p\subseteq Q^p_\sigma\}_{p\in\pi,\sigma\in\Sigma}$, then, for each $p \in \pi$, $\sigma \in \Sigma$,

$$P^p_\sigma;\bigcup_{j=1}^{M_p}\{R(p,j,0);X(p,j,1);\ldots;X(p,j,K_{p,j});R(p,j,K_{p,j})\}\subseteq Q^p_\sigma.$$

(α) If $K_{p,j}=0$, then $P^p_\sigma;R(p,j,0)\subseteq Q^p_\sigma$, by (4.4).

(β) Let $K_{p,j}\geq 0$. Then

$$\begin{aligned}
&P^p_\sigma;R(p,j,0);X(p,j,1);\ldots;R(p,j,K_{p,j}) = && (4.3)\\
&P^{h(p,j,1)}_{\sigma(p,j,1)};X(p,j,1);\ldots;R(p,j,K_{p,j})\subseteq && (*)\\
&Q^{h(p,j,1)}_{\sigma(p,j,1)};R(p,j,1);\ldots;R(p,j,K_{p,j})\subseteq\ldots\subseteq && \\
&Q^{h(p,j,K_{p,j})}_{\sigma(p,j,K_{p,j})};R(p,j,K_{p,j})\subseteq Q^p_\sigma && (4.4)
\end{aligned}$$

(α) and (β) together imply that we have proved the second assumption of the induction theorem. Thus, we conclude that $\psi[P_p/X_p]_{p\epsilon\pi}$ holds, i.e., that $\{P^p_\sigma;P_p\subseteq Q^p_\sigma\}_{p\epsilon\pi,\sigma\epsilon\Sigma}$. From this, (4.8) follows and the proof of part 2, and, therefore, of the regularization theorem is completed.

4.3. The inductive assertion theorem

Let ρ be an index set, and let $D=\{P_r,S_r\}_{r\epsilon\rho}$ be a closed declaration scheme over $\{P_r\}_{r\epsilon\rho}\cup\mathcal{A}\cup\mathcal{C}$ (i.e., the S_r contain no occurrences of an $X\epsilon\mathcal{X}$ or of a $p\epsilon\mathcal{A}_p$). As in subsection 4.2, we assume that each S_r is of the special form (4.1), (4.2). (We replace from now on $p(\pi)$ by $r(\rho)$ in order to avoid conflicts of notation.)

Let K_r, $M_{r,j}$ and Σ be as above, and let $\Sigma_{0+}=\{(r,j,k)\mid r\epsilon\rho,\ 1\leq j\leq K_r,\ 0\leq k\leq M_{r,j}\}$.

Let $\mathcal{E}=\{p^r_\sigma,q^r_\sigma\}_{r\epsilon\rho,\sigma\epsilon\Sigma}$ be a collection of predicate symbols, contained in A_p.

We define the *inductive assertion pattern* $A[D,\mathcal{E}]$ with respect to the declaration scheme D and the collection of predicate symbols $\mathcal{E}$ as follows: First, for each $\sigma\epsilon\Sigma$, $(r,j,k)\epsilon\Sigma_{0+}$ we define $\alpha^\sigma_{(r,j,k)}[D,\mathcal{E}]$ by putting

1. If $K_{r,j} = 0$, then

$$\alpha^{\sigma}_{(r,j,0)}[D,E] \quad = p^{r}_{\sigma};R(r,j,0) \subseteq R(r,j,0);q^{r}_{\sigma}.$$

2. If $K_{r,j} > 0$, then

   a. $\alpha^{\sigma}_{(r,j,0)}[D,E] \quad = p^{r}_{\sigma};R(r,j,0) \subseteq R(r,j,0);p^{h(r,j,1)}_{\sigma(r,j,1)}$.

   b. For $1 \leq k \leq K_{r,j}-1$

   $$\alpha^{\sigma}_{(r,j,k)}[D,E] \quad = q^{h(r,j,k)}_{\sigma(r,j,k)};R(r,j,k) \subseteq R(r,j,k);p^{h(r,j,k+1)}_{\sigma(r,j,k+1)}.$$

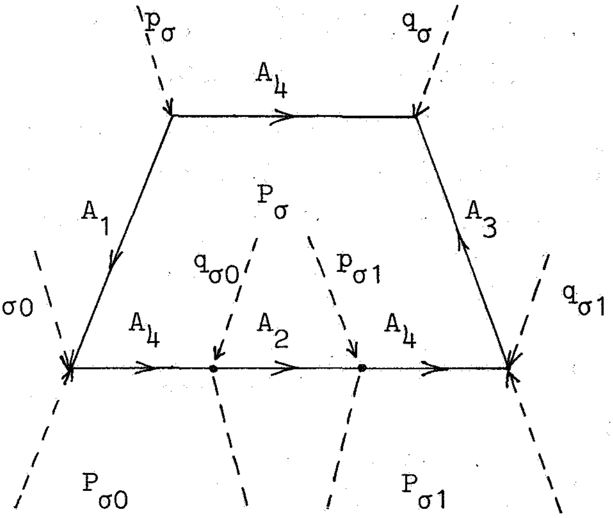
   c. $\alpha^{\sigma}_{(r,j,K(r,j))}[D,E] = q^{h(r,j,K(r,j))}_{\sigma(r,j,K(r,j))};R(r,j,K(r,j)) \subseteq R(r,j,K(r,j));q^{r}_{\sigma}$.

Now let $A[D,E]= \{\alpha^{\sigma}_{(r,j,k)}\}_{\sigma\in\Sigma,(r,j,k)\in\Sigma_{0+}}$. Then, as we shall see, $A[D,E]$ provides the solution to our problem.

Example: Let D be the declaration scheme $\{P \Leftarrow A_1;P;A_2;P;A_3 \cup A_4\}$. We have for $A[D,E]$:

$$\left\{\begin{array}{l} p_{\sigma};A_1 \subseteq A_1;p_{\sigma 0} \\ q_{\sigma 0};A_2 \subseteq A_2;p_{\sigma 1} \\ q_{\sigma 1};A_3 \subseteq A_3;q_{\sigma} \\ p_{\sigma};A_4 \subseteq A_4;q_{\sigma} \end{array}\right\}_{\sigma\in\Sigma=\{0,1\}^*}$$

The following picture, referring to an inner call of P with history σ ($\sigma 0$ and $\sigma 1$) may illustrate the idea:



First we prove a lemma.

LEMMA 4.1.

Let D,$\mathcal{E}$ be as above. Let $\{T_r\}_{r\in\rho}$ be arbitrary statement schemes over $\mathcal{P} \cup \mathcal{A} \cup \mathcal{C}$. Then the following two assertions are equivalent:

1. $\{T_r \subseteq P_r\}_{r\in\rho}$

2. $\mathcal{A}[D,\mathcal{E}] \vdash \{p^r_\sigma;T_r \subseteq T_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma}$.

PROOF

1. 1 $\Longrightarrow$ 2. First we prove this for the special case that $\{T_r = P_r\}_{r\in\rho}$. By Scott's induction rule (theorem 3.2) it is sufficient to prove that

$$(4.9) \qquad \mathcal{A}[D,\mathcal{E}],\ \{p^r_\sigma;X_r \subseteq X_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma} \vdash \{p^r_\sigma;\tilde{S}_r \subseteq \tilde{S}_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma},$$

where, as usual, $\tilde{S}_r$ results from S_r by substituting for each $r \in \rho$, P_r for X_r. In order to prove (4.9) it is sufficient to show that, for each j, $1 \leq j \leq M_r$, we can infer from its assumptions that

$$p^r_\sigma;R(r,j,0);X(r,j,1);\ldots;R(r,j,K(r,j)) \subseteq$$
$$R(r,j,0);X(r,j,1);\ldots;R(r,j,K(r,j));q^r_\sigma.$$

By A[D,E], $p^r_\sigma;R(r,j,0) \subseteq R(r,j,0);p^{h(r,j,1)}_{\sigma(r,j,1)}$. By the assumption in (4.9) on the X_r, and the definition of the h-function, $p^{h(r,j,1)}_{\sigma(r,j,1)};X(r,j,1) \subseteq$ $\subseteq X(r,j,1);q^{h(r,j,1)}_{\sigma(r,j,1)}$. Repeating this argument, which is straightforward from the definition of A[D,E], the desired result follows, i.e., the proof of 1 $\Longrightarrow$ 2 for the case that $\{T_r = P_r\}_{r\in\rho}$ holds, is completed. Next assume that $\{T_r \subseteq P_r\}_{r\in\rho}$. Then, clearly,

$$A[D,E] \vdash \{p^r_\sigma;T_r \subseteq p^r_\sigma;P_r \subseteq P_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma}.$$

Also, $\{p^r_\sigma;T_r \subseteq T_r\}_{r\in\rho,\sigma\in\Sigma}$. Since $\{q^r_\sigma \subseteq E\}_{r\in\rho,\sigma\in\Sigma}$, the desired conclusion $A[D,E] \vdash \{p^r_\sigma;T_r \subseteq T_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma}$ follows.

2. 2 $\Longrightarrow$ 1. We have to show: For all interpretations c, $\{c(T_r) \subseteq c(P_r)\}_{r\in\rho}$. Let c be an arbitrary interpretation, and let c_0 be its initial interpretation, i.e., $c_0 = c \mid R$. Since none of the p^r_σ,q^r_σ occurs in T_r or S_r, we can extend c_0 to c'_0 without causing any change in c, as follows: Let V be the domain of c_0, and let, for each $r \in \rho$, x_r be an arbitrary element of V. We put $(x,y) \in c'_0(p^r_\sigma)$ iff x = y, and, moreover, $(x_r,x) \in$ $\in c(P^r_\sigma)$, where P^r_σ is the procedure defined in (4.3). Similarly, $(x,y) \in c'_0(q^r_\sigma)$ iff x = y, and, moreover, $(x_r,x) \in c(Q^r_\sigma)$, with Q^r_σ defined as in (4.4). Let c' be the extension of c'_0. About this c' we can now prove: Each element of $\{\alpha^\sigma_{(r,j,k)}\}_{\sigma\in\Sigma,(r,j,k)\in\Sigma_{0+}}$ satisfies c'. In fact, our extensive preparations are rewarded here, since the proof is direct from the definitions of A[D,E], P^r_σ, and Q^r_σ. By the assumption of the lemma we have, since A[D,E] satisfies c', that $\{p^r_\sigma;T_r \subseteq T_r;q^r_\sigma\}_{r\in\rho,\sigma\in\Sigma}$, also satisfies c'. In particular, for $\sigma = \Sigma$, we have that $\{p^r_\varepsilon;T_r \subseteq T_r;q^r_\varepsilon\}_{r\in\rho}$ satisfies c'. Next, we use that $(x,y) \in c'_0(p^r_\varepsilon)$ iff $x = x_r$, which follows from $P^r_\varepsilon = E$, and that $(x,y) \in c'_0(q^r_\varepsilon)$ iff $(x_r,x) \in c(Q^r_\varepsilon) = c(P_\varepsilon)$, which follows from $P_\varepsilon = Q^r_\varepsilon$. Thus, we have shown that: if $x = x_r$, and $xc(T_r)y$, then $xc(P_r)y$. Since x_r was an arbitrary element of V, we conclude that $\{c(T_r) \subseteq c(P_r)\}_{r\in\rho}$.
   This completes the proof of lemma 4.1.

It is now easy to give the proof of the inductive assertion theorem:

THEOREM 4.2 (the inductive assertion theorem)

Let D, $\mathcal{E}$ be as above, and let $\{T_r\}_{r\epsilon\rho}$ be fixed points of the statement schemes $\{S_r\}_{r\epsilon\rho}$, i.e., let

$$\{\tilde{S}_r[T_r/X_r]_{r\epsilon\rho} = S_r\}_{r\epsilon\rho}.$$

Then the following two assertions are equivalent

1. $\{\tilde{S}_r[T'_r/X_r]_{r\epsilon\rho} \subseteq T'_r\}_{r\epsilon\rho} \vdash \{T_r \subseteq T'_r\}_{r\epsilon\rho}$,

   i.e., the $\{T_r\}_{r\epsilon\rho}$ are *minimal* fixed points.

2. $A[D,\mathcal{E}] \vdash \{p^r_\sigma;T_r \subseteq T_r;q^r_\sigma\}_{r\epsilon\rho,\sigma\epsilon\Sigma}$,

PROOF

1. 1 $\Longrightarrow$ 2. If the $\{T_r\}_{r\epsilon\rho}$ are minimal fixed points, then, by Corollary 3.1, $\{T_r = P_r\}_{r\epsilon\rho}$, and the result follows by lemma 4.1.
2. 2 $\Longrightarrow$ 1. By lemma 4.1, if 2 holds, then $\{T_r \subseteq P_r\}_{r\epsilon\rho}$ follows. Thus, the $\{T_r\}_{r\epsilon\rho}$ are fixed points which are included in, and thus equal to, the minimal fixed points $\{P_r\}_{r\epsilon\rho}$.

APPENDIX: DERIVATIVES AND TRACES

In [6], Hitchcock and Park introduce the notion of *derivative* of a program scheme, and use it in the development of a technique for giving proofs of termination of programs.

Roughly speaking, the derivative relates states at successive nested calls of a procedure to each other. Therefore, the question arose as to the clarification of the relationship between this notion and our "tracing" constructs P_σ. In this appendix we state (without proof) a theorem settling this question.

First we repeat the definition of [6], somewhat reformulated for the present purpose:

Let T be a statement scheme over $\{P_r\}_{r\epsilon\rho} \cup \mathcal{R}$. Then $\frac{\partial T}{\partial P_r}$ is defined inductively as follows:

a. If $T \in \mathcal{R}$, then $\frac{\partial T}{\partial P_r} = \Omega$.

b. If $T \in \{P_r\}_{r\epsilon\rho}$, then $\frac{\partial P_{r_1}}{\partial P_r} = E, \quad r_1 = r$

$= \Omega, \quad r_1 \neq r.$

c. If $T = T_1;T_2$, then $\frac{\partial T}{\partial P_r} = \frac{\partial T_1}{\partial P_r} \cup T_1;\frac{\partial T_2}{\partial P_r}$.

d. If $T = T_1 \cup T_2$, then $\frac{\partial T}{\partial P_r} = \frac{\partial T_1}{\partial P_r} \cup \frac{\partial T_2}{\partial P_r}$.

Example: $\frac{\partial(A_1;P;A_2;P;A_3)}{\partial P} = A_1 \cup A_1;P;A_2.$

The theorem relating derivatives and traces now follows:

THEOREM A

Let $\{P_r,S_r\}_{r\epsilon\rho}$ be a declaration scheme.

Let Δ_{r_1,r_2} be defined by: $\Delta_{r_1,r_2} = E, \quad r_1 = r_2,$

$= \Omega, \quad r_1 \neq r_2.$

Let, for $r_1,r_2 \in \rho$, $d_{r_1}P_{r_2}$ be a new procedure "symbol", with declaration:

$$\{d_{r_1} P_{r_2} \Longleftarrow \Delta_{r_1,r_2} \cup \bigcup_{r\epsilon\rho} (\frac{\partial S_{r_2}}{\partial P_r}; d_{r_1} P_r)\}_{\substack{r_1\epsilon\rho \\ r_2\epsilon\rho}}.$$

(The expression $\bigcup\limits_{r\epsilon\rho} \ldots$ is only seemingly of infinite length, since only for finitely many r, $\frac{\partial S_{r_2}}{\partial P_r} \neq \Omega$.)

Let $\emptyset$ be the empty set.

Let $\Sigma_{r_1,r_2} = \{w \in \Sigma_0^* \mid \sigma_{r_1} \overset{*}{\Longrightarrow} w\} \cup$ (if $r_1 = r_2$ then $\{\varepsilon\}$ else $\emptyset$) (cf. the definition of page 27).

Then

$$\{d_{r_1} P_{r_2} = \bigcup_{\sigma\epsilon\Sigma_{r_2,r_1}} P_\sigma^{r_1}\}_{r_1\epsilon\rho, r_2\epsilon\rho}.$$

REFERENCES

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

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

[3] De Bakker, J.W. & W.P. de Roever, A calculus for recursive program schemes, to appear in Proc. IRIA Symposium on Automata, Formal Languages and Programming, North-Holland, Amsterdam.

[4] Bekić, H., Definable operations in general algebra, and the theory of automata and flowcharts, Report IBM Laboratory Vienna (1969).

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

[6] Hitchcock, P. & D. Park, Induction rules and proofs of termination, to appear in Proc. IRIA Symposium on Automata, Formal Languages and Programming, North-Holland, Amsterdam.

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

[8] Manna, Z. & J.M. Cadiou, Recursive definitions of partial functions and their computations, in Proc. of an ACM Conference on proving assertions about programs, pp. 58-65, ACM (1972).

[9] Manna, Z., S. Ness & J. Vuillemin, Inductive methods for proving properties of programs, in. Proc. of an ACM Conference on proving assertions about programs, pp. 27-50, ACM (1972).

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

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

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

[13] Morris Jr,. J.H., Another recursion induction principle, C. ACM, 14, pp.351-354 (1971).

[14] Milner, R., Implementation and applications of Scott's logic for computable functions, in Proc. of an ACM Conference on proving assertions about programs, pp. 1-6, ACM (1972).

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

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

[17] Scott, D., in Minutes of the fourth meeting of the IFIP Working Group 2.2 on Formal Language Description Languages, Essex University (1969).

[18] Tarski, A., A lattice theoretical fixpoint theorem and its applications, Pacific J. of Math., 5, 285-309 (1955).