stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 80/77 MAART

K.R. APT & J.W. DE BAKKER

SEMANTICS AND PROOF THEORY OF PASCAL PROCEDURES

Preprint

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).

AMS(MOS) subject classification scheme (1970): 68A05

ACM-Computing Reviews-categories: 5.24

Semantics and proof theory of PASCAL procedures*)

by

K.R. Apt & J.W. de Bakker

ABSTRACT

A simple sublanguage of PASCAL containing simple and subscripted variables, assignment, sequential composition, conditionals, procedure calls, and declarations of simple variables, of arrays and of (recursive) procedures, is considered. The method of denotational semantics is used to define the meaning of the language constructs and a corresponding proof system in the style of Hoare is proposed. The main new feature is a rigorous treatment of recursive procedures with the parameter mechanisms of call-by-value and call-by-variable, of locality and of scope problems.

KEY WORDS & PHRASES: *denotational semantics, systems of mutually recursive procedures, call-by-value, call-by-variable, locality, program correctness, Hoare's proof system*

---

*) This report will be submitted for publication elsewhere.

1. INTRODUCTION

Our paper is devoted to an investigation of (recursive) procedures with the parameter mechanisms of call-by-value and call-by-variable (the FORTRAN call-by-reference) as occurring in the language PASCAL. We use the method of denotational semantics (SCOTT & STRACHEY [15], MILNE & STRACHEY [13]) and propose proof rules in the style of HOARE [7].

Our analysis is presented within the framework of a simple sublanguage of PASCAL, containing simple and subscripted variables, a few simple kinds of expressions, assignment, sequential composition, conditionals, declarations of simple variables, of arrays and of procedures, and procedure calls. It turns out that an adequate treatment of procedures in this setting - i.e., a treatment which describes their meaning both exactly as in the PASCAL report and with sufficient mathematical rigour - is already quite complicated, reason why we have organised the presentation in a number of stages. In section 2, we present syntax and semantics of a very simple language with expressions (these remain the same throughout the paper) and as statements only assignment statements, together with sequential composition and conditionals. In section 3 we add parameterless recursive procedures - which are by now reasonably well understood - and introduce our notation for their least fixed point semantics. Section 4 brings the full story on procedures with parameters called-by-value and called-by-variable, together with a treatment of declarations and the ensuing scope problems. In section 5 we finally give the corresponding proof theory.

We now list some of the more difficult issues encountered in our investigation, and indicate the tools used to overcome them.

Semantics

- scope problems (e.g. as described in the ALGOL 60 report, sections 4.7.3.2 and 4.7.3.3): these have been tackled essentially by a systematic and careful use of *substitution* (in the sense as used e.g. in predicate logic).
- declarations: these are dealt with through more or less standard use of environments (dynamically varying partial functions from variables to addresses) and of a somewhat similar construct assigning meaning to procedure variables.
- procedures with parameters: these are dealt with by suitable combination of the least fixed point technique and the technique of "syntactic application" by which

a procedure body together with the actuals of the call are mapped to a new piece of program text.

Proof theory

- assignment to subscripted variables (arising through subscripted variables used as actual parameter called-by-variable): an extension of Hoare's assignment axiom, using a new definition of substitution for a *subscripted* variable (details are given in DE BAKKER [3]), is proposed.
- procedures with parameters: an extension of Scott's induction rule using the above mentioned "syntactic application" is proposed.

Apart from PASCAL concepts which we consider more or less "orthogonal" to the concept of procedure (goto's, data types etc. - e.g., we do not treat index types for arrays), there are two major omissions in the paper, viz. function designators in expressions, and procedures and functions as parameters. The first omission is mainly motivated by expected complications in the proof theory, the second by anticipated problems in the semantics.

The issues studied in our paper have been discussed already by several authors. Without aiming at completeness, we mention the following: In the work as exemplified by MANNA & VUILLEMIN [12] call-by-value and call-by-name-like parameter mechanisms are investigated, but only in so far as the *order* of evaluation of the actuals is concerned. Problems raised by scope considerations, or by subscripted variables as actuals, are not dealt with. Another contribution to the subject is the recently published book by DONAHUE [5]. In his approach (as e.g. in the older one of LAUER [11]) occurrences of global variables in procedure bodies are not allowed. An attempt at removing this restriction in HOARE & WIRTH [9] leads to problems concerning an inadequacy of the proposed proof rules. In the papers by COOK [4] and GORELICK [6], global variables are treated incorrectly (see a discussion of these issues in [5], p.41 and 139). HOARE [8], COOK [4], DONAHUE [5], and IGARASHI et al. [10] impose restrictions on the actual parameters in procedure calls- in particular, their syntax does not allow Jensen's device. In DONAHUE [5] and IGARASHI et al. [10], subscripted variables cannot be passed as actuals called-by-variable. In HOARE [8], COOK [4] and GORELICK [6], only call-by-name is allowed.

Finally, let us mention what we see as the main contributions of our paper.

a. A treatment of procedures which allows *unrestricted* use of both globals and of parameters called-by-value and called-by-variable.
b. The use of a "proof-theory-oriented" denotational semantics, which considerably facilitates justification of the proof rules. (Syntactic application and substitution play an important role here.)
c. An extension of computational (or Scott's) induction which we believe to be new. (Some partial steps towards these goals have already been made in our [1]. However,

syntactic application was used there only in rudimentary form, and neither *local* nor *systems* of procedure declarations were allowed.)

Acknowledgements

R. Milne and P. Mosses made some helpful comments on a previous version of this paper.

2. EXPRESSIONS. SYNTAX AND SEMANTICS OF S_0 (assignment, composition, conditionals)

We use a slight variant of BNF which should be self-explanatory.

Expressions

As starting point we take the classes *SV* of *simple variables* with x,y,z,u,... as typical elements, *AV* of *array variables* with a,b,... as typical elements, and *C* of *integer constants* with m,n,... as typical elements. For later use we assume these sets to be well-ordered. We then define

IV (*integer variables*) with typical elements v,w,...
v::= x|a[s] (i.e., a variable v is simple or subscripted)

IE (*integer expressions*) with typical elements s,t,...
t::= n|v|t_1+t_2|...| <u>if</u> p <u>then</u> t_1 <u>else</u> t_2 <u>fi</u>
(other operations may be added)

BE (*boolean expressions*) with typical elements p,q,...
p::= <u>true</u>|<u>false</u>|t_1=t_2|...|$p_1 \supset p_2$|...| <u>if</u> p <u>then</u> p_1 <u>else</u> p_2 <u>fi</u>
(other operations may be added).

Statements

The class of (as yet very simple) statements S_0 is defined as:

S_0 (*statements*) with typical elements S,S_1,...
S::= v:=t|S_1;S_2| <u>if</u> p <u>then</u> S_1 <u>else</u> S_2 <u>fi</u>

Throughout the paper we do not bother about syntactic ambiguities which may be remedied by suitable addition of parentheses. "≡" is used to denote syntactic identity. *Meaning* is attributed to expressions and statements in the following manner: Let *I* be the set of integers with ν as a typical element, and let {T,F} be the set of truth values. In the subsequent development (section 4) variables are mapped to integers via an intermediate step of *addresses*. In the present simplified situation we do not yet introduce these, but map variables directly to integers. There is a slight complication for subscripted variables, dealt with as follows: Let *Var* = *SV* ∪ (*AV*×*I*) be the set of elements x,... ∈ *SV* united with the set of elements (a,μ),... ∈ *AV* × *I*. We then define a *store* σ as a mapping from *Var* to *I*. Let Σ be the set of all stores. The meaning of an expression or statement is defined with respect to a store in the following way: we introduce the mappings

L_0: $IV \to (\Sigma \to Var)$ (*left-hand-value* of an integer variable),
R_0: $IE \to (\Sigma \to I)$ (*right-hand-value* of an integer expression),
T_0: $BE \to (\Sigma \to \{T,F\})$ (*value* of a boolean expression), and
M_0: $S_0 \to (\Sigma \to \Sigma)$ (*value* or *meaning* of a statement).

First we need the notion of *variant* of a store. Let ζ,ζ' be typical elements of *Var*. Then $\sigma\{\nu/\zeta\}(\zeta')$ is defined by:

$$\sigma\{\nu/\zeta\}(\zeta') = \nu, \quad \text{if } \zeta = \zeta',$$
$$\sigma\{\nu/\zeta\}(\zeta') = \sigma(\zeta'), \quad \text{if } \zeta \neq \zeta'.$$

We now define:

$L_0(x)(\sigma) = x$, $L_0(a[s])(\sigma) = (a,R_0(s)(\sigma))$,

$R_0(n)(\sigma) = \nu$ (where ν is the integer denoted by the integer constant n),

$R_0(v)(\sigma) = \sigma(L_0(v)(\sigma))$, $R_0(t_1+t_2)(\sigma) = plus\ (R_0(t_1)(\sigma), R_0(t_2)(\sigma))$,...,

$$R_0\ (\underline{\text{if}}\ p\ \underline{\text{then}}\ t_1\ \underline{\text{else}}\ t_2\ \underline{\text{fi}})(\sigma) = \begin{cases} R_0(t_1)(\sigma) & \text{if } T_0(p)(\sigma) = T, \\ R_0(t_2)(\sigma) & \text{if } T_0(p)(\sigma) = F. \end{cases}$$

Example: $R_0(a[a[1+1]])(\sigma) = \sigma(a,\sigma((a,2)))$.

$T_0(\underline{\text{true}})(\sigma) = T$, $T_0(\underline{\text{false}})(\sigma) = F$, $T_0(t_1 = t_2)(\sigma) = equal(R_0(t_1)(\sigma), R_0(t_2)(\sigma))$,...,

$T_0(p_1 \supset p_2)(\sigma) = (T_0(p_1)(\sigma) \Longrightarrow T_0(p_2)(\sigma))$, (with " $\Longrightarrow$" the usual implication between truth values),...,

$M_0(v:=t)(\sigma) = \sigma\{R_0(t)(\sigma)/L_0(v)(\sigma)\}$,

$M_0(S_1;S_2)(\sigma) = M_0(S_2)(M_0(S_1)(\sigma))$,

$$M_0\ (\underline{\text{if}}\ p\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}})(\sigma) = \begin{cases} M_0(S_1)(\sigma) & \text{if } T_0(p)(\sigma) = T, \\ M_0(S_2)(\sigma) & \text{if } T_0(p)(\sigma) = F. \end{cases}$$

(These definitions are all straightforward and presented like this to get the reader somewhat accustomed to the notation before confronting him with the main issues of this paper.)

3. SYNTAX AND SEMANTICS OF S_1 (S_0 with parameterless procedures added)

Syntax.

IV, *IE* and *BE* are as before. Let *PV* be a class of *procedure variables* with P,Q,... as typical elements. S_1 with $S,S_1,\ldots$ as typical elements is defined as

$$S ::= v:=t \mid S_1;S_2 \mid \underline{\text{if}}\ p\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}} \mid P$$

Let E with $E, E_1 \ldots$ as typical elements be the set of *procedure declarations* given by

$$E ::= P <== S \mid E_1, E_2 \mid \qquad \text{(note that this allows an empty E)}$$

where it is required that in each declaration $P_1 <== S_1, \ldots, P_n <== S_n$, $P_i \neq P_j$ for $i \neq j$, $1 \leq i,j \leq n$. ($P <== S$ abbreviates the parameterless procedure declaration procedure P;S as e.g. in ALGOL 60. Note that we do not yet allow procedure declarations within the procedure bodies: this restriction is lifted in section 4.)

Semantics.

Σ and I are as before, and so are $L_1 (= L_0)$, $R_1 (= R_0)$ and $T_1 (= T_0)$. Let us consider a statement $S \in S_1$. Its meaning with respect to a system of declarations E will now be defined as a *partial* function ξ from states to states (i.e., as an element in the set $\Sigma \xrightarrow[\text{part}]{} \Sigma$ with typical elements $\xi, \eta, \ldots$). E.g., if $S \equiv P$, we want its meaning with respect to $P <== P$ to be the nowhere defined function. We allow the possibility that one or more procedure variables occurring in S are undeclared in E, and we use an element θ in the class of mappings $\Theta_1 : PV \to (\Sigma \xrightarrow[\text{part}]{} \Sigma)$ to assign (some arbitrary) meaning to such undeclared procedure variables. Before presenting the definitions, we introduce the following notation (compare the $\sigma\{\ldots\}$ formalism): For each $\theta \in \Theta_1$, $P \in PV$ and $\eta \in \Sigma \xrightarrow[\text{part}]{} \Sigma$, we define $\theta\{\eta/P\}$ by:

$$\theta\{\eta/P\}(P) = \eta, \quad \theta\{\eta/P\}(Q) = \theta(Q) \quad \text{for all } Q \neq P.$$

Moreover, we order $\Sigma \xrightarrow[\text{part}]{} \Sigma$ by putting: $\eta \subseteq \eta'$ iff, for all σ, either $\eta(\sigma)$ is undefined, or $\eta(\sigma) = \eta'(\sigma)$.

We now define $M_1 : E \times S_1 \to (\Theta_1 \to (\Sigma \xrightarrow[\text{part}]{} \Sigma))$ (writing $M_1(E|S)(\theta)(\sigma)$ in order to avoid certain ambiguities below).

First we give the easy cases:

$$M_1(E|v:=t)(\theta)(\sigma) = \sigma\{R_1(t)(\sigma)/L_1(v)(\sigma)\}$$

$$M_1(E|S_1;S_2)(\theta)(\sigma) = M_1(E|S_2)(\theta)(M_1(E|S_1)(\theta)(\sigma))$$

$$M_1(E|\underline{\text{if}}\ p\ \underline{\text{then}}\ S_1\ \underline{\text{else}}\ S_2\ \underline{\text{fi}})(\theta)(\sigma) = \begin{cases} M_1(E|S_1)(\theta)(\sigma) & \text{if } T(p)(\sigma) = T \\ M_1(E|S_2)(\theta)(\sigma) & \text{if } T(p)(\sigma) = F. \end{cases}$$

$M_1(E|P)$ is defined through the following process (which is, in fact, nothing but the well-known least-fixed-point semantics, justified on the basis of the operational meaning of procedures e.g. in [2]):

Let $E \equiv P_1 <== S_1, \ldots, P_n <== S_n$.

$$M_1(E|P)(\theta)(\sigma) = \theta\{\xi_1/P_1\} \ldots \{\xi_n/P_n\}(P)(\sigma)$$

where, for $j = 1,\ldots,n$, ξ_j is obtained in the following way: Let, for $i = 1,\ldots,n$, the operator $\Phi_i: (\Sigma \xrightarrow[\text{part}]{} \Sigma)^n \to (\Sigma \xrightarrow[\text{part}]{} \Sigma)$ be given by

$$\Phi_i(\eta_1,\ldots,\eta_n) = M(|S_i)(\theta\{\eta_1/P_1\} \ldots \{\eta_n/P_n\}).$$

Let the ordering "$\subseteq$" on $\Sigma \xrightarrow[\text{part}]{} \Sigma$ be extended in the natural way to $(\Sigma \xrightarrow[\text{part}]{} \Sigma)^n$. Then $\xi_j = \mu_j[\Phi_1,\ldots,\Phi_n]$, $j = 1,\ldots,n$ where $\mu_j[\Phi_1,\ldots,\Phi_n]$ is the j-th component of the least fixed point of the n-tuple of operators $\Phi_1,\ldots,\Phi_n$ (i.e., $\mu_j[\Phi_1,\ldots,\Phi_n] = \xi_j$, $j = 1,\ldots,n$ iff $(\xi_1,\ldots,\xi_n) = \text{glb}\{(\eta_1,\ldots,\eta_n)|\Phi_i(\eta_1,\ldots,\eta_n) = \eta_i,\ i = 1,\ldots,n\}$.)

Observe that if P is undeclared in E, $M_1(E|P)(\theta) = \theta(P)$ is some arbitrary η.

4. SYNTAX AND SEMANTICS OF S_2 (S_0 with declarations and full procedures)

Syntax.

Expressions are as before. We introduce the class of statements S (dropping, as everywhere below, the index 2) using auxiliary classes R^1, R^2, R^3, E and PB defined as follows:

S::=	$R^1 \mid \underline{\text{var}}\ x;R^1$	($S \in S$)
R^1::=	$R^2 \mid \underline{\text{array}}\ a;R^2$	($R^1 \in R^1$)
R^2::=	$R^3 \mid E;R^3$	($R^2 \in R^2$)
R^3::=	$v:=t \mid R^3_1;R^3_2 \mid \underline{\text{if}}\ p\ \underline{\text{then}}\ R^3_1\ \underline{\text{else}}\ R^3_2\ \underline{\text{fi}} \mid P(t,v) \mid \underline{\text{begin}}\ S\ \underline{\text{end}}$	($R^3 \in R^3$)
E::=	$P<==B \mid E_1,E_2 \mid$ (restricted as in section 3)	($E \in E$)
B::=	$<\underline{\text{val}}\ x;\underline{\text{var}}\ y \mid S>$ where $x \neq y$	($B \in PB$).

Remarks.

(i) The construct $<\underline{\text{val}}\ x;\underline{\text{var}}\ y|S>$ denotes a procedure body with x as formal value parameter and y as formal variable parameter.

(ii) All considerations of this and the next section can be trivally extended to the case of, possibly empty, lists of variable declarations, array declarations, or formal parameters (we use this fact tacitly in the definition of syntatic application below).

(iii) Separate treatment of the begin S end case, being trivial, is always omitted in the sequel.

S is essentially a subset of PASCAL (apart from the begin S end construct which ensures that the outcome of syntatic application (see below) is an element of S).

Semantics.

As important extension of the previous cases we now use the notion of an *environment* mapping variables to an infinite well-ordered set A of addresses. Stores are then mappings from A to I. More precisely, we define a class *Env*, with typical elements $\varepsilon, \varepsilon_1, \ldots$, of *partial* functions: $Var \xrightarrow[\text{part}]{} A$ satisfying certain restrictions (each ε is 1 - 1; specification of other (technical) restrictions is omitted for simplicity's sake). Recall that $Var = SV \cup (AV \times I)$. Environments are used in the treatment of (variable and array) declarations which, in turn, play a role in the semantics of procedure calls. Their use implies a refinement in the definitions of L_1, R_1 and T_1, which now are of the following type:

$L: IV \xrightarrow[\text{part}]{} (Env \times \Sigma \to A)$,

$R: IE \xrightarrow[\text{part}]{} (Env \times \Sigma \to I)$,

$T: BE \xrightarrow[\text{part}]{} (Env \times \Sigma \to \{T,F\})$.

We give a few clauses of their definitions:

$L(x)(\varepsilon,\sigma) = \varepsilon(x)$,

$L(a[s])(\varepsilon,\sigma) = \varepsilon(a,R(s)(\varepsilon,\sigma))$,

$R(v)(\varepsilon,\sigma) = \sigma(L(v)(\varepsilon,\sigma)),\ldots$,

$T(t_1=t_2)(\varepsilon,\sigma) = equal(R(t_1)(\varepsilon,\sigma),R(t_2)(\varepsilon,\sigma)),\ldots$.

Let $H = IE \times IV \to (Env \times \Sigma \xrightarrow[\text{part}]{} \Sigma)$ and let $\eta,\eta',\eta_1,\ldots$ be typical elements of H. We order H by putting $\eta \subseteq \eta'$ iff $\forall t \forall v(\eta(t,v) \subseteq \eta'(t,v))$. Θ is now defined as $PV \to H$. For each $\theta \in \Theta$, $P \in PV$ and $\eta \in H$ we define $\theta\{\eta/P\}$ as before.

The *meaning* M of $S \in S$ with respect to $E \in E$ and $\theta \in \Theta$ in environment ε and store σ yields a new store σ', i.e.

$$M: E \times S \to (\Theta \to (Env \times \Sigma \xrightarrow[\text{part}]{} \Sigma)).$$

In our approach the treatment of declarations and procedures calls requires a number of preparations, viz. a new notation, and the introduction two major tools:

(a) *Notation.* For any $\varepsilon \in Env$, $y \in SV$ such that $y \notin dom(\varepsilon)$ and $\alpha \in A$ such that $\alpha \notin range(\varepsilon)$, we write $\varepsilon \cup \langle y,\alpha\rangle$ for the extension of ε yielding α when applied to y. Similarly we write $\varepsilon \cup \langle\langle a,\nu\rangle,\alpha_\nu\rangle_{\nu \in I}$ for the extension of ε yielding α_ν when applied to $\langle a,\nu\rangle (\nu \in I)$.

(b) *Substitution.* A careful definition of substitution of an integer variable v for a (simple) variable x in a statement S, written as $S[v/x]$, and, mutatis mutandis, of $S[b/a]$ and $S[Q/P]$ plays a major part in our approach. Substitution is - as we see it - the proper tool for dealing with scope problems, and, moreover, it is used in an essential manner in the treatment of parameter passing. We do not present its full

definition, but give some representative cases.

An occurence of a variable in a statement can be bound or free (e.g., an occurence of x in S is bound whenever it is within a substatement of S of the form var x;R^1, or <val x;var y|S_1>, or <val z;var x|S_1>. Mutatis mutandis, we define the other cases).

$(w:=t)[v/x] \equiv w[v/x]:= t[v/x]$ (where substitution in expressions is straightforward),

$(R_1^3;R_2^3)[v/x] \equiv R_1^3[v/x];R_2^3[v/x]$, $P(t,w)[v/x] \equiv P(t[v/x], w[v/x])$,

(var y;R^1)[v/x]

$\equiv$ var y;R^1, if x $\equiv$ y

$\equiv$ var y;R^1[v/x], if x $\not\equiv$ y and y not free in v,

$\equiv$ var y';R^1[y'/y][v/x], if x $\not\equiv$ y and y free in v, where y' is the first variable $\in SV$ such that y' $\not\equiv$ x and y' not free in R^1 or V.

$(E;R^3)[v/x] \equiv E[v/x];R^3[v/x]$, $(E_1,E_2)[v/x] \equiv E_1[v/x],E_2[v/x]$, $(P<=B)[v/x] \equiv P<=B[v/x]$,

<val x; var y|S>[v/z]

$\equiv$ <val x; var y|S>, if z $\equiv$ x or z $\equiv$ y,

$\equiv$ <val x; var y|S[v/z]>, if z $\not\equiv$ x and z $\not\equiv$ y and x and y not free in v,

$\equiv$ similar as above, otherwise

(c) *Syntactic application.* The next main idea of this section is the notion of "syntactic application": For each procedure body B we define its application B[t,v] to the actuals t (corresponding to the formal value parameter) and v (corresponding to the formal variable parameter) as a *syntactic* operation (i.e., as an operation yielding a piece of text) as follows:

(<val x; var y|S>)[t,z] $\equiv$ var u;u:=t;begin S[u/x][z/y]end,

(<val x; var y|S>)[t,a[s]] $\equiv$ var u_1,u_2;u_1:=t;u_2:=s;begin S[u_1/x][a[u_2]/y]end,

where it is required that u is the first variable $\not\equiv$ x,y and not free in S,t or z (analogously for u_1,u_2). Observe that

(i) this definition implies that the actual value parameter t is indeed evaluated before execution of S;

(ii) the precaution with the fresh u is necessary since a definition like var x;x:= t; ... might give a clash between the local x and possible occurences of x in the actual t (cf. ALGOL 60 report, 4.7.3.2);

(iii) the two possibilities for the actual variable parameter v are
- v z, a simple variable. Call-by-variable then coincideswith the ALGOL 60 call-by-name.

- $v \equiv a[s]$, a subscripted variable. Then s is evaluated (and stored in u_2) before execution of S.

We now define M as follows:

$$M(E|v:=t)(\theta)(\varepsilon,\sigma) = \sigma\{R(t)(\varepsilon,\sigma)/L(v)(\varepsilon,\sigma)\}$$

$$M(E|R_1^3;R_2^3)(\theta)(\varepsilon,\sigma) = M(E|R_2^3)(\theta)(\varepsilon,M(E|R_1^3)(\theta)(\varepsilon,\sigma))$$

$$M(E|\underline{\text{if}}\ p\ \underline{\text{then}}\ R_1^3\ \underline{\text{else}}\ R_2^3\ \underline{\text{fi}})(\theta)(\varepsilon,\sigma) = \begin{cases} M(E|R_1^3)(\theta)(\varepsilon,\sigma) & \text{if } T(p)(\varepsilon,\sigma) = T \\ M(E|R_2^3)(\theta)(\varepsilon,\sigma) & \text{if } T(p)(\varepsilon,\sigma) = F \end{cases}$$

$M(E|\underline{\text{var}}\ x;R^1)(\theta)(\varepsilon,\sigma) = M(E|R^1[y/x])(\theta)(\varepsilon \cup \langle y,\alpha\rangle,\sigma)$,
where y is the first variable $\in SV$ not in dom(ε), and α the first address not in range(ε)

$M(E|\underline{\text{array}}\ a;R^2)(\theta)(\varepsilon,\sigma) = M(E|R^2[b/a])(\theta)(\varepsilon \cup \langle\langle b,\nu\rangle,\alpha_\nu\rangle_{\nu\in I},\sigma)$
where b is the first array variable such that no $\langle b,\nu\rangle$ is in dom(ε), and where the α_ν are chosen in some (unspecified but) unique way from $A \setminus$ range(ε)

$$M(E|P_1<==B_1,\ldots,P_n<==B_n;R^3)(\theta)(\varepsilon,\sigma) =$$
$$M(E,Q_1<==B_1[Q_j/P_j]_{j=1}^n,\ldots,Q_n<==B_n[Q_j/P_j]_{j=1}^n|R^3[Q_j/P_j]_{j=1}^n)(\theta)(\varepsilon,\sigma)$$
where the $Q_1,\ldots,Q_n$ are the first variables $\in PV$ such that for each j, $j = 1,\ldots,n$, Q_j does not occur in E, $P_1<==B_1,\ldots,P_n<==B_n$ or R^3.

$M(E|P(t,v))(\theta)(\varepsilon,\sigma) = \theta\{\xi_1/P_1\}\ldots\{\xi_n/P_n\}(P)(t,v)(\varepsilon,\sigma)$
where $E \equiv \langle P_i<==B_i\rangle_{i=1}^n$ and for $j = 1,\ldots,n$, ξ_j is obtained in the following way: Let, for $i = 1,\ldots,n$, the operator $\Phi_i: H^n \to H$ be given by $\Phi_i(\eta_1,\ldots,\eta_n) = \lambda t'\cdot\lambda v'\cdot M(|B_i[t',v'])(\theta\{\eta_1/P_1\}\ldots\{\eta_n/P_n\})$. Then $\xi_j = \mu_j[\Phi_1,\ldots,\Phi_n]$, for $j = 1,\ldots,n$.

It is always assumed that ε is defined for all simple and array variables which are free in E or S.

Summarizing, declarations are dealt with by suitably extending the environment (ε or E (which may just as well be viewed as environment for the procedure variables)), with the precaution of always choosing fresh variables. Procedure calls are treated through a combination of least fixed point techniques with a syntactic device for dealing with parameter mechanisms.

5. PROOF THEORY

The proof theory is given in the style of HOARE [7], but it extends previous approaches in that we

- present an axiom for assignment to *subscripted* variables as well (a detailed description of this is contained in DE BAKKER [3])
- present a proof rule for (recursive) procedure calls which embodies an adequate treatment of parameter passing
- present proof rules for declarations which properly deal with scope problems (for variable declarations our rule is taken from HOARE [8])

We describe only the kernel of the system. Certain rules which are standard in this type of deductive system are omitted and extension of BE with quantifiers together with the appropriate extension of the function T is left to the reader.

An *atomic correctness formula* is a construct of the form $\{p\}S\{q\}$, $p,q \in BE$, $S \in S$. Arbitrary atomic correctness formulae are denoted by $\gamma, \gamma_1, \ldots$ and Γ denotes a finite set of such atomic formulae. A *correctness formula* is a construct of the form $\langle E|\Gamma\rangle$. It is *not* required that each P occurring in some $\{p\}S\{q\}$ from Γ be declared in E (it will be, however, the case with all provable $\langle E|\Gamma\rangle$). Proof rules are of two forms:

$$\frac{\langle E_1|\Gamma_1\rangle}{\langle E_2|\Gamma_2\rangle}, \quad \text{and} \quad \frac{\langle E|\Gamma_1\rangle \rightarrow \langle E|\Gamma_2\rangle}{\langle E|\Gamma_3\rangle}.$$

First we define the *validity* of a correctness formula $\langle E|\Gamma\rangle$. Let $E \equiv \langle P_i <= B_i \rangle_{i=1}^{n}$. Let (as before) $H = (IE \times IV) \rightarrow (Env \times \Sigma \xrightarrow[part]{} \Sigma)$, and let H^E be the set of all E-*variable invariant* η in H, i.e., η satisfies, for each t,v,x,ε,α,σ:

(i) $\eta(t[y'/x], v[y'/x])(\varepsilon \cup \langle y', \alpha\rangle, \sigma) = \eta(t[y''/x], v[y''/x])(\varepsilon \cup \langle y'', \alpha\rangle, \sigma)$
where y', y'' are variables $\in SV$ not occurring free in t, v or in any procedure body in E

(ii) $\eta(t,v)(\varepsilon,\sigma)(\varepsilon(y)) = \sigma(\varepsilon(y))$
where y is a variable $\in SV$ which does not occur free in t,v or in any procedure body in E

The first condition is needed for (the soundness of) the substitution rule whereas the second is needed for (the validity of) the invariance axiom. Observe that for each $i = 1,\ldots,n$ $\lambda t \cdot \lambda v \cdot M(E|P_i(t,v)) \in H^E$

We define

(i) $M(\langle E|\{p\}S\{q\}\rangle)(\theta)$ holds iff for all ε defined on all the free variables of E, p, S and q and for all σ, we have: if $T(p)(\varepsilon,\sigma)$ and for some σ', $M(E|S)(\theta)(\varepsilon,\sigma) = \sigma'$ then $T(q)(\varepsilon,\sigma')$.

(ii) $M(\langle E|\Gamma\rangle)(\theta)$ holds iff $M(\langle E|\gamma\rangle)(\theta)$ holds for each $\gamma \in \Gamma$.

(iii) $\langle E|\Gamma\rangle$ is *valid* iff $M(\langle E|\Gamma\rangle)(\theta)$ holds for each $\theta \in PV \to H^E$

(iv) $\dfrac{\langle E_1|\Gamma_1\rangle}{\langle E_2|\Gamma_2\rangle}$ is *sound* iff for all $\theta \in PV \to H^{E_1} \cap H^{E_2}$, $M(\langle E_1|\Gamma_1\rangle)(\theta)$ implies $M(\langle E_2|\Gamma_2\rangle)(\theta)$

(v) $\dfrac{\langle E|\Gamma_1\rangle \to \langle E|\Gamma_2\rangle}{\langle E|\Gamma_3\rangle}$ is *sound* iff soundness of $\dfrac{\langle E|\Gamma_1\rangle}{\langle E|\Gamma_2\rangle}$ implies validity of $\langle E|\Gamma_3\rangle$.

The following axioms and proof rules are proposed:

Assignment $\langle E|\{p[t/v]\}v:=t\{p\}\rangle$.

This is like Hoare's axiom, but it extends it since substitution for a *subscripted* variable is also covered. We present the central clause from its definition:
$a[s][r/a[t] \equiv \underline{\text{if}}\ s[r/a[t]] = t\ \underline{\text{then}}\ r\ \underline{\text{else}}\ a[s[r/a[t]]]\ \underline{\text{fi}}$.

Composition, conditionals. As usual and omitted.

Declarations $\dfrac{\langle E|\{p\}R^1[y/x]\{q\}\rangle}{\langle E|\{p\}\ \underline{\text{var}}\ x;R^1\{q\}\rangle}$,

where y is some variable not occurring free in E,p,R^1 or q.
(The array case is similar and omitted.)
For procedure declarations:

$$\frac{\langle E, Q_1 <= B_1[Q_j/P_j]_{j=1}^n, \ldots, Q_n <= B_n[Q_j/P_j]_{j=1}^n \mid \{p\}R^3[Q_j/P_j]_{j=1}^n\{q\}\rangle}{\langle E \mid \{p\}P_1 <= B_1, \ldots, P_n <= B_n; R^3\{q\}\rangle}$$

where the Q_j are completely fresh (formal definition of this omitted).

Remark. The rule for procedure declarations motivates the carrying along of E in a correctness formula for the purpose of recording the procedure bodies.

Procedure calls. We proceed in four stages for didactic reasons.

1. (Only one non-recursive procedure declaration.) Assume the declaration P <= B. If B contains no occurrences of P, then we have the following simple proof rule:

$$\frac{\langle E, P <= B \mid \{p\}B[t,v]\{q\}\rangle}{\langle E, P <= B \mid \{p\}P(t,v)\{q\}\rangle}$$

(Note that, in the recursive case this rule is still sound, though presumably not very useful.)

2. (Only one parameterless procedure declaration.) Assume the declaration P <= S. We then have

$$\langle E, P \Leftarrow S \mid \{p_0\}P'\{q_0\},\ldots,\{p_n\}P'\{q_n\}\rangle$$

$$\rightarrow$$

$$\frac{\langle E, P \Leftarrow S \mid \{p_0\}S'\{q_0\},\{p_1\}S'\{q_1\},\ldots,\{p_n\}S'\{q_n\}\rangle}{\langle E, P \Leftarrow S \mid \{p_0\}P\{q_0\},\{p_1\}P\{q_1\},\ldots,\{p_n\}P\{q_n\}\rangle}$$

where $S' \equiv S[P'/P]$ and P' is a fresh variable.

(This is a "parallel" version of Scott's well-known induction rule for parameterless procedures. The structure of this version anticipates stage 3.)

3. (Only one procedure declaration.) Assume the declaration $P \Leftarrow B$ with $B \equiv \langle \underline{val}\ x; \underline{var}\ y \mid S\rangle$ in which

$$S \equiv \ldots P(t_1,v_1)\ldots\sim\ldots\ P(t_i,v_i)\ldots\sim\ldots\ P(t_n,v_n)\ldots\ .$$

We then have

$$\langle E, P \Leftarrow B \mid \{p_1\}P'(t_1,v_1)\{q_1\},\ldots,\{p_n\}P'(t_n,v_n)\{q_n\}\rangle$$

$$\rightarrow$$

$$\frac{\langle E, P \Leftarrow B \mid \{p_0\}B'[t_0,v_0]\{q_0\},\{p_1\}B'[t_1,v_1]\{q_1\},\ldots,\{p_n\}B'[t_n,v_n]\{q_n\}\rangle}{\langle E, P \Leftarrow B \mid \{p_0\}P(t_0,v_0)\{q_0\}\rangle}$$

where $B' \equiv B[P'/P]$ and P' is a fresh variable.

Observe that the antecedent of the premise of the rule consists of correctness formulae concerning the *inner* recursive calls $P(t_i,v_i)$, $i = 1,\ldots,n$ with appropriately chosen "inner" assertions p_i, q_i, $i = 1,\ldots,n$, whereas the conclusion of the rule concerns the *outer* call $P(t_0,v_0)$, with outer assertions p_0, q_0. Observe also that there is no point in including $\{p_0\}P(t_0,v_0)\{q_0\}$ in the antecedent of the premise, nor of $\{p_i\}P(t_i,v_i)\{q_i\}$, $i = 1,\ldots,n$ in the conclusion of the rule.

4. (A system of procedure declarations.) We now consider the system of procedure declarations $E' \equiv \langle P_i \Leftarrow B_i\rangle_{i=1}^{n}$. We use index-triples (j,k,h) to denote the h-th occurrence $(1 \leq h)$ of the k-th procedure variable $(1 \leq k \leq n)$ in the j-th procedure body $(1 \leq j \leq n)$, with $h \leq M_{j,k}$ where $M_{j,k}$ denotes the number of occurrences of P_k in B_j. Of course, P_k always occurs in B_j in the form $P_k(t,v)$ for some $(t,v) = (t_{(j,k,h)}, v_{(j,k,h)})$ $(1 \leq h \leq M_{j,k})$. Let

$$J = \{(j,k,h) \mid 1 \leq j \leq n,\ 1 \leq k \leq n,\ 1 \leq h \leq M_{j,k}\}$$

and let k_0 be an element of the set $\{1,\ldots,n\}$.

$$\frac{\begin{array}{l}\langle E,E' \mid \{p_{(j,k,h)}\}P'_k(t_{(j,k,h)},v_{(j,k,h)})\{q_{(j,k,h)}\}_{\substack{k=1,\ldots,n\\(j,k,h)\in J}}\rangle \\ \rightarrow \\ \langle E,E' \mid \{p_{k_0}\}B'_{k_0}[t_{k_0},v_{k_0}]\{q_{k_0}\}, \\ \quad \{p_{(j,k,h)}\}B'_k[t_{(j,k,h)},v_{(j,k,h)}]\{q_{(j,k,h)}\}_{\substack{k=1,\ldots,n\\(j,k,h)\in J}}\rangle\end{array}}{\langle E,E' \mid \{p_{k_0}\}P_{k_0}(t_{k_0},v_{k_0})\{q_{k_0}\}\rangle}$$

where for $k = 1,\ldots,n$ $B'_k \equiv B_k[P'_i/P_i]_{i=1}^n$ and $P'_1,\ldots,P'_k$ are fresh variables.

Remarks.

1. The P'_k are not declared in E, E', hence, in the premise of the rule the meaning of P'_k is an arbitrary $\eta_k \in H^{E,E'}$. However, the soundness definition implies universal quantification over these η, as follows from the role of θ.

2. Again, the conclusion is to show $\langle E,E' \mid \{p_{k_0}\}P_{k_0}(t_{k_0},v_{k_0})\{q_{k_0}\}\rangle$ for some k_0 $1 \leq k_0 \leq n$, where the $P_{k_0}(t_{k_0},v_{k_0})$ is the outer call, and the $P_k(t_{(j,k,h)},v_{(j,k,h)})$ are inner recursive calls, occurring within the B_j, and having to satisfy the premise for (suitably chosen) inner assertions $p_{(j,k,h)}$ and $q_{(j,k,h)}$.

3. Our rule is an extended version of Scott's induction rule (see SCOTT & DE BAKKER [14]) - its soundness can be shown by a more or less standard appeal to the continuity of the operators involved in the least fixed point definition. Observe that in our approach *each* call can be treated (by applying some appropriate instance of the rule), which is not the case with the approach originated by HOARE [8] and followed by IGARASHI et. al. [10], GORELICK [6] and DONAHUE [5]. As we understand it, they use Scott's rule only for actual parameters (which happen to be) identical to the corresponding formals and then apply some substitution process to deal with other actuals. However, they are then enforced to impose various restrictions on these actuals.

Two types of rules and one final axiom are needed to allow meaningful application of the procedure rule.

Substitution

$$\frac{\langle E \mid \{p\}P(t,v)\{q\}\rangle}{\langle E \mid \{p[w/x]\}P(t[w/x],v[w/x])\{q[w/x]\}\rangle}$$

where

(i) x does not occur free in E.

(ii) either $w \equiv z$ or $w \equiv a[z]$ for some $z \in SV$ which does not occur free in E,p,t,v or q.

Remark. This rule presents a limited treatment of substitution; stronger versions are possible (see COOK [4] and GORELICK [6]).

Extension

$$\frac{\langle E \mid \{p\}S\{q\}\rangle}{\langle E,E' \mid \{p\}S\{q\}\rangle}$$

where no procedure variable declared in E' occurs free in E or S.

Invariance

$$\langle E \mid \{p\}P(t,v)\{p\}\rangle$$

where p does not have free simple variables which occur free in E, t or v.

Remark. This axiom is taken from GORELICK [6].

We illustrate the use of the procedure rule by means of an example.

Example. Let S be the following statement:

if x = 0 then a[0]:=1 else x:=x-1; P(x,a[x]); y:= (x+1)·a[x] fi

and let B be <val x; var y|S>. For any integer expression t we want to prove

$$\langle P \Leftarrow B \mid \{t \geq 0\}P(t,a[t])\ \{\forall u(u \leq t \supset a[u]=u!)\}\rangle.$$

We have to make use of the procedure rule (from stage 3) with appropriately chosen inner assertions p_1 and q_1 for the inner call P(x,a[x]). Observe that we have not only to verify that the inner call does its subtask but also that it does not change the value of x. For this reason we choose $x \geq 0 \wedge z = x$ as p_1 and $\forall u(u \leq x \supset a[u]=u!) \wedge z = x$ as q_1 where u and z are fresh variables. By the procedure rule it is now sufficient to prove

$$\langle P \Leftarrow B \mid \{x \geq 0 \wedge z = x\}P'(x,a[x])\{\forall u(u \leq x \supset a[u]=u!) \wedge z = x\}\rangle$$

$$\rightarrow$$

$$\langle P \Leftarrow B \mid \{t \geq 0\}B'[t,a[t]]\{\forall u(u \leq t \supset a[u] = u!)\},$$

$$\{x \geq 0 \wedge z = x\}B'[x,a[x]]\{\forall u(u \leq x \supset a[u] = u!) \wedge z = x\}\rangle,$$

where $B' \equiv B[P'/P]$ and P' is a fresh variable.

Because of space limits we leave the details to the reader. The proof uses the substitution rule and the invariance axiom.

REFERENCES

[1] APT, K.R. & J.W. DE BAKKER, *Exercises in denotational semantics*, in: Proc. 5th Symposium on Mathematical Foundations of Computer Science (A. Mazurkiewicz, ed.), pp. 1-11, Lecture Notes in Computer Science 45, Springer (1976).

[2] BAKKER, J.W. DE, *Least fixed points revisited*, Theoretical Computer Science, 2, pp. 155-181 (1976).

[3] BAKKER, J.W. DE, *Correctness proofs for assignment statements*, Report IW 55/76 Mathematisch Centrum (1976).

[4] COOK, S.A., *Axiomatic and interpretive semantics for an ALGOL fragment*, Technical Report no. 79, University of Toronto (1975).

[5] DONAHUE, J.E., *Complementary definitions of programming language semantics*, Lecture Notes in Computer Science 42, Springer (1976).

[6] GORELICK, G.A., *A complete axiomatic system for proving assertions about recursive and non-recursive programs*, Technical Report no. 75, University of Toronto (1975).

[7] HOARE, C.A.R., *An axiomatic basis for programming language constructs*, C.ACM 12, pp. 576-580 (1969).

[8] HOARE, C.A.R., *Procedures and parameters: an axiomatic approach*, in: Symp. on Semantics or Algorithmic Languages, Lecture Notes in Mathematics 188 (E. Engeler, ed.) pp. 102-116, Springer (1971).

[9] HOARE, C.A.R. & N. WIRTH, *An axiomatic definition of the programming language PASCAL*, Acta Inf. 2, pp. 335-355 (1973).

[10] IGARASHI, S., R.L. LONDON & D.C. LUCKHAM, *Axiomatic program verification I: A logical basis and its implementation*, Acta Inf. 4, pp. 145-182 (1975).

[11] LAUER, P.W., *Consistent formal theories of the semantics of programming languages*, Report TR 25 121, IBM Laboratory, Vienna (1971).

[12] MANNA, Z. & J. VUILLEMIN, *Fixpoint approach to the theory of the computation*, C.ACM 15, pp. 528-536 (1972).

[13] MILNE, R. & C. STRACHEY, *A theory of programming language semantics*, Chapman and Hall, London and Wiley, New York (1976).

[14] SCOTT, D. & J.W. DE BAKKER, *A theory of programs*, unpublished memo (1969).

[15] SCOTT, D. & C. STRACHEY, *Towards a mathematical semantics for computer languages*, in: Proc. of the Symp. on Computers and Automata (J. Fox, ed.) pp. 19-46, Polytechnic Inst. of Brooklyn (1971).