MATHEMATICAL LOGIC APPLIED TO THE SEMANTICS OF COMPUTER PROGRAMS

by

Edward Anthony Ashcroft

Ph. D. Thesis
submitted to the
Faculty of Engineering
University of London

June 1970

Abstract:

A definition of the semantics of a programming language is considered to be some method of formally describing the computations of programs written in that language. For such a definition to be more satisfactory than an actual interpreter or compiler, its formal aspects must give it certain advantages such as generality or descriptive ability. In addition, it is desirable that formal proofs of correctness of compilers or properties of programs be possible using such a definition.

New techniques for proving properties of programs have been developed from initial work by Floyd. These techniques relate programs to formulae of mathematical logic.

To allow such proof techniques to be used directly from the formal definitions of programs, we consider defining the semantics of programs by formulae of mathematical logic. We develop criteria which can reasonably be said to ensure that such definitions are intuitively 'adequate'.

It is then shown that such adequate definitions are closely related to the formulae used in the logical proof techniques. Such definitions can therefore be used to prove various properties of the programs they define.

Two examples are given of such definitions. Firstly for a functional language, like a restricted form of Lisp. Secondly for a large subset of Algol 60. With this latter definition it may be possible to prove properties of practical Algol programs.

Contents

Section 1: INTRODUCTION

The usual method of specifying the semantics of a high-level programming language is by a programming manual. However, in cases where detailed knowledge is required of the execution of involved or unusual programs, such manuals are frequently inadequate. The classic example of such a situation is of course when a compiler is to be written for the language in question. So it was not long after the development of the first high level languages that the need was felt for some more rigorous description of the semantics of programming languages than could be supplied by natural language texts.

One of the first steps in this direction was made by McCarthy [12] with his definition of Micro-Algol. The definition took the form of an abstract interpreter, and most of the subsequent work in language definition has gone into the development of interpreter definitions, for example, the PL/I definition [6]. Thus these definitions are able, in principle, to answer the question: "What is the result of executing this program for these input data?" Since this question can also be answered by actual interpreters (or compilers), the abstract interpreters must have other useful properties such as generality, and descriptive ability, and be formal enough to allow proofs of properties of programs or of the correctness of compilers.

However, even for the very simple Algol-like languages, such interpreter definitions did not lend themselves to simple proofs of the correctness of compilers [13, 14] or of properties of programs.

An alternative approach was that of Landin [5] who took λ-calculus as a model programming language, and defined other languages by mapping their programs into λ-expressions. This could be called a compiler definition approach. In common with all such approaches, it suffers from the drawback that the intuitive meaning of the more complicated constructions in the original language is obscured by the translation, or else such constructions are just not allowed. However, with recent work on models of λ-calculus by Scott [17, 18], it is anticipated that there will be renewed interest in compiler definitions.

A third approach was proposed by Floyd [4], namely that of a definition using logic. In that paper he presented a method of proving the partial correctness (q.v) of a program based on the 'verification conditions' of the various statements in the program. The rules for obtaining the verification conditions for the various statements constituted a 'semantic definition' of the programming language; and it was desirable that the verification conditions so obtained had certain properties ('consistency' and 'completeness') related to the usual intuitive notions about the execution of the statements concerned. This approach thus gives a definition specifically designed for proving properties of programs.

The method of proving partial correctness was taken up, and made more formal, by Manna [7] but the verification conditions, now expressed as a formula in predicate calculus, were no longer considered as embodying the semantics of the program. Instead the formula was shown to be related to an interpreter definition of the semantics of the program. As the formalization of partial correctness has been extended to more complicated systems [8, 9, 11, 2] it has become more and more difficult to give an

interpreter definition for each system and rigorously establish its relationship with the predicate calculus formula. Instead an intuitive description of the semantics has been given, with an informal argument to exhibit the desired relationship to the predicate calculus formula. However, the last step, of calling the formula itself a semantic definition, has not been made, even though it has been shown by Manna [10] that a formula formalizing partial correctness of programs can be used to formalize all the usual properties of programs such as termination, equivalence, etc.

As a variant of the logical approach, Burstall [3] has recently given a description of the semantics of a large subset of Algol in first order logic. From the formula embodying the semantics one can derive, using the rules of logic, a sequence of sentences describing the computation of the program. As a definition of semantics this is intuitively more appealing than a set of verification conditions, simply because of its similarity to an interpreter definition. It also has the advantages of descriptive ability and it can be used to prove properties of programs.

In this work we develop the notion of intuitively 'adequate' logical definitions of programs and languages, namely those of an interpreter-like nature. Burstall's definition will be seen to be 'adequate' in this sense. We also show that logical program definitions are closely related to Manna's formalizations of partial correctness and therefore can be used to prove properties of programs. In addition, for the type of languages considered by Floyd, we show that language definitions by complete and consistent verification conditions are adequate logical language definitions in our sense.

We then give examples of language definitions, first for a functional language, and then for a large subset of Algol. The definitions in both cases can be used to obtain formalizations of partial correctness of programs.

Throughout this work we assume some familiarity with elementary first order mathematical logic. Standard notation is used for logical connectives, quantifiers, etc. as in, for example, 'Introduction to Mathematical Logic' by E. Mendelson.

Section 2: LOGICAL PROGRAM DEFINITION

Adequacy of a logical program definition

It seems clear that for a definition of semantics of a program to be generally acceptable it must be an interpreter; for any given input, the definition must specify the result, if any, of executing the program. Moreover, it must do this in as mechanical a way as possible. For a definition written in first-order logic, this suggests that the definition should take the form of a set of axioms, and that the result of the program should be derivable from the axioms in as mechanical a manner as possible, e.g. using a mechanical theorem prover.

The combination of definitional axioms + theorem-prover will then act like program + interpreter. The main difference will be that the theorem-prover will 'compute' with expressions formed from symbols representing the constants and primitive operations of the program, whereas the interpreter deals with real data. Since the interpreter must make decisions based on its data at various points in the computation using the primitive tests of the program, the same feature will be necessary in the theorem-prover. To give the theorem-prover this ability, it could be given additional axioms that specify the domain of data, and the primitive operations and tests, sufficiently for it to be able to deduce the truth or falsity of any test when applied to symbolic expressions representing data objects. Alternatively, it could be considered to be interactive, interrogating the 'outside world' whenever it needed to make such a test.

In either case, if we consider the domain of data and the primitive tests and operations on it as a 'relational structure', $\mathcal{I}$ [*] say, then this method of deduction we will call $\mathcal{I}$-relative deduction, and denote it by $\vdash^{\mathcal{I}}$. For proper axioms K, $\vdash^{\mathcal{I}}_{K}$ denotes $\mathcal{I}$-relative deduction from K.

For any term τ_c constructed from symbols representing the constants and basic operations of a program (called a constant term), the structure $\mathcal{I}$ will determine an associated value in $|\mathcal{I}|$ (the domain of $\mathcal{I}$), which will be denoted by $\mathcal{I}[\tau_c]$.

With this notation, we can state the requirements for a semantic definition to be intuitively adequate.

A program $P_{\mathcal{I}}$, which requires values for n inputs and gives m output values, can be considered as a partial function $P_{\mathcal{I}}: |\mathcal{I}|^n \to |\mathcal{I}|^m$, i.e., for inputs $\xi = (\xi_1 \cdots \xi_n) \in |\mathcal{I}|^n$, $P_{\mathcal{I}}(\xi)$ denotes the output values, if any.

The relation $A_{P_{\mathcal{I}}}$ on $|\mathcal{I}|^{n+m}$, called the graph of $P_{\mathcal{I}}$, is simply the relation defined by $P_{\mathcal{I}}$, i.e., for $\zeta = (\zeta_1 \cdots \zeta_m) \in |\mathcal{I}|^m$

$$P_{\mathcal{I}}(\xi) = \zeta \Leftrightarrow A_{P_{\mathcal{I}}}(\xi, \zeta) .$$

The aim of the axioms + theorem-prover (T.P) system is simply to specify this relation $A_{P_{\mathcal{I}}}$.

---

[*] Most of the constants, functions and tests in $\mathcal{I}$ will correspond to symbols occurring in programs. However, some may be implicit in programs, such as the operations for updating and referencing arrays.

To supply input values $\xi \in |\mathcal{I}|^n$ to the axioms + T.P system, there are constants $(k_1 \ldots k_n) = k$ in the definitional axioms, and we expand the structure $\mathcal{I}$ to a structure $\mathcal{I}_k^{\xi}$. That is, constants k are assigned values ξ by this expanded structure.

In order to indicate when the result $P_{\mathcal{I}}(\xi)$ of the program has been derived, the axioms contain a distinguished n+m-ary predicate symbol, ϕ say, which is not interpreted by $\mathcal{I}$.

For clarity, a set of axioms W, constituting a definition of $P_{\mathcal{I}}$, will therefore be written as $W_P(k,\phi)$ */.

We can now state the following definition:

A) The logical program definition condition

$W_P(k,\phi)$ is an adequate logical definition of $P_{\mathcal{I}}$ if it satisfies the following condition:

$\forall \xi \in |\mathcal{I}|^n$, $\forall \zeta \in |\mathcal{I}|^m$:

$A_{P_{\mathcal{I}}}(\xi,\zeta)$ if and only if

there exist constant terms $(\tau_1,\ldots,\tau_m) = \tau$

s.t. $\mathcal{I}[\tau] = \zeta$ and $\vdash^{\mathcal{I}_k^{\xi}}_{W_P(k,\phi)} \phi(k,\tau)$

i.e., $\phi(k,\tau)$ is derived exactly in the case where τ denotes the result of the program for input ξ.

---

*/ We shall call the symbols denoting the constants and basic operations and tests of $P_{\mathcal{I}}$ the 'formal basis' of $P_{\mathcal{I}}$. Note that $\mathcal{I}$ is a structure just for the formal basis symbols. In general, apart from the formal basis symbols, the extra constants k and the predicate ϕ, $W_P(k,\phi)$ may contain other constant, function and predicate symbols, and variables.

We are going to relate adequate logical definitions to formalizations of partial correctness (Manna) and to consistent and complete 'verification conditions' (Floyd). To do this, the concept of $\mathcal{I}$-relative deduction must be related to $\mathcal{I}$-relative validity. This is the purpose of the following subsection.

A diversion into model theory

As for programs, we shall call the constant, function and predicate symbols, to which a structure $\mathcal{I}$ assigns meaning, the formal basis of $\mathcal{I}$. Any closed well-formed-formula (first-order), constructed from the formal basis symbols together with individual variables, will be true or false for $\mathcal{I}$ in the usual way.*/

Any w.f.f. Γ constructed from symbols of some basis, together with variables, is said to be valid, denoted $\models\Gamma$, if the closure of Γ is true in all structures for the basis symbols.

If a wff Γ contains 'extra' symbols $\{\Sigma\}$ not in the formal basis of some structure $\mathcal{I}$, then an 'expansion of $\mathcal{I}$ to include $\{\Sigma\}$' is a structure identical to $\mathcal{I}$ except that it also assigns meaning to the extra symbols in $\{\Sigma\}$. Then Γ is said to be $\mathcal{I}$-relative valid, denoted $\models^{\mathcal{I}}\Gamma$, if its closure is true in all expansions of $\mathcal{I}$ to include $\{\Sigma\}$. Analogously to Godel's Completeness Theorem, we would like to show that

$$\models^{\mathcal{I}}\Gamma \Leftrightarrow \vdash^{\mathcal{I}}\Gamma .$$

*/ Some familiarity with first order logic and elementary model theory is assumed. The notation used in this subsection is that used in Schoenfield [6].

Unfortunately this does not hold for general Γ . However, it will be shown below that it holds for most Γ of interest.

For any structure $\mathcal{I}$ with formal basis L (of constant, function, predicate symbols, and equality) we can define a substructure $\bar{\mathcal{I}}$ such that $|\bar{\mathcal{I}}|$ is the smallest subset of $|\mathcal{I}|$ containing the elements corresponding to the constants of L , which is closed under the operations corresponding to the functions of L . That is, $|\bar{\mathcal{I}}|$ contains just those elements of $|\mathcal{I}|$ that correspond to terms in the formal basis symbols.

The restricted form of the completeness theorem for $\mathcal{I}$-relative logic can be stated.

$\mathcal{I}$-relative Completeness Theorem

For any existential formula*/ A containing only predicate symbols that are not in the formal basis of $\mathcal{I}$

$$\vdash^{\mathcal{I}} A \Leftrightarrow \models^{\bar{\mathcal{I}}} A \quad .$$

Proof.

=> Let $\mathcal{I}$ be over the formal basis L , and the extra predicate symbols in A be $\{R\}$. Denote by $D'(\mathcal{I})$ the set of variable free formulas in L that are true in $\mathcal{I}$.

Clearly from the intuitive description of $\mathcal{I}$-relative deduction given previously,

---

*/ An existential formula is a closed formula that contains only existential quantifiers when put in prenex normal form.

$$\vdash^{\mathcal{J}} A \Leftrightarrow \underset{D'(\mathcal{J})}{\vdash} A$$

$$\Leftrightarrow \underset{D'(\mathcal{J})}{\models} A \quad \text{by Godel Completeness Theorem.}$$

Since every expansion of $\bar{\mathcal{J}}$ to include $\{R\}$ must be a model of $D'(\mathcal{J})$, trivially

$$\vdash^{\mathcal{J}} A \Rightarrow \models^{\bar{\mathcal{J}}} A \ .$$

$\Leftarrow$ With the notation of the first half of the proof, assume

$$\models^{\bar{\mathcal{J}}} A \not\Rightarrow \underset{D'(\mathcal{J})}{\models} A \ .$$

Then there is some structure $\mathcal{C}$ which is a model of $D'(\mathcal{J})$, but in which A is not true. Since A is closed, and existential, the universal*/ formula $\neg A$ is true in $\mathcal{C}$.

We shall show that the substructure $\bar{\mathcal{C}}$ of $\mathcal{C}$ is isomorphic to an expansion of $\bar{\mathcal{J}}$, and hence contradicts the above assumption. First we define a bijective mapping $\phi : |\bar{\mathcal{C}}| \to |\bar{\mathcal{J}}|$. By definition, every element i of $|\bar{\mathcal{C}}|$ is the value in $\mathcal{C}$ of some term τ_i constructed from the basis symbols of $\mathcal{C}$. Note that τ_i is also constructed from the basis symbols of $\mathcal{J}$, since we have introduced no extra function symbols or constants.

We define $\phi(i) = \mathcal{J}[\tau_i]$ for all $i \in |\bar{\mathcal{C}}|$. Now for elements i_1, i_2 of $|\bar{\mathcal{C}}|$

$$\phi(i_1) = \phi(i_2) \Rightarrow [\tau_{i_1} = \tau_{i_2} \text{ is a formula in } D'(\mathcal{J})]$$

$$\Rightarrow i_1 = i_2$$

$\therefore$ mapping ϕ is injective.

---

*/ A universal formula is closed.

Since every element of $|\bar{\mathcal{I}}|$ is the value in $\mathcal{I}$ of some term constructed from the basis symbols, the mapping ϕ is surjective. Hence

i) ϕ is bijective.

It remains to check that the structure on $|\bar{\mathcal{C}}|$ is the same as an expansion of the structure on $|\bar{\mathcal{I}}|$.

For n-ary function symbol f in the formal basis of $\mathcal{C}$, let $f_{\mathcal{C}}$ and $f_{\mathcal{I}}$ denote the corresponding functions in $\mathcal{C}$ and $\mathcal{I}$. Then for elements $i_1 \ldots i_n \in |\bar{\mathcal{C}}|$, corresponding to terms $\tau_{i_1} \ldots \tau_{i_n}$

$$\begin{aligned}\phi(f_{\mathcal{C}}(i_1,\ldots,i_n)) &= \phi(\mathcal{C}[f(\tau_{i_1},\ldots,\tau_{i_n})]) \\ &= \mathcal{I}[f(\tau_{i_1},\ldots,\tau_{i_n})] \\ &= f_{\mathcal{I}}(\phi(i_1),\ldots,\phi(i_n)) \quad , \text{ i.e.,}\end{aligned}$$

ii) For all function symbols f in the formal basis of $\bar{\mathcal{C}}$, and elements $i_1,\ldots,i_n$ of $|\bar{\mathcal{C}}|$

$$\phi(f_{\bar{\mathcal{C}}}(i_1,\ldots,i_n)) = f_{\bar{\mathcal{I}}}(\phi(i_1),\ldots,\phi(i_n)) .$$

Then for every n-ary predicate symbol p in the formal basis of $\mathcal{I}$ (i.e., not in $\{R\}$)

$$\begin{aligned}p_{\mathcal{C}}(i_1,\ldots,i_n) &\Leftrightarrow [p(\tau_{i_1},\ldots,\tau_{i_n}) \text{ is in } D'(\mathcal{I})] \\ &\Leftrightarrow p_{\mathcal{I}}(\phi(i_1),\ldots,\phi(i_n)) .\end{aligned}$$

Also for every n-ary predicate symbol p in {P} we expand $\bar{\mathfrak{I}}$ (to $\bar{\mathfrak{I}}'$) to include {R} in such a way that

$$p_{\mathcal{C}}(i_1,\ldots,i_n) \Leftrightarrow p_{\bar{\mathfrak{I}}'}(\phi(i_1),\ldots,\phi(i_n))$$

$\therefore$ iii) For every predicate symbol p in the formal basis of $\bar{\mathcal{C}}$

$$p_{\bar{\mathcal{C}}}(i_1,\ldots,i_n) \Leftrightarrow p_{\bar{\mathfrak{I}}'}(\phi(i_1),\ldots,\phi(i_n))$$

$\therefore$ by i) ii) and iii) $\bar{\mathcal{C}}$ is isomorphic to an expansion of $\bar{\mathfrak{I}}$.

Now by the Łos'-Tarki theorem (Schoenfield, § 5.2), every universal formula that is true in some structure $\mathcal{C}$ is true in all substructures of $\mathcal{C}$ */

$\therefore \neg A$ is true in $\bar{\mathcal{C}}$ and, by the isomorphism, also in some expansion of $\bar{\mathfrak{I}}$. But by assumption, A is true in all expansions of $\bar{\mathfrak{I}}$.

$\therefore$ Contradiction.

Q.E.D.

The following corollary is obvious.

Corollary. If K is a universal formula, and A is an existential formula and both contain only predicate symbols that are not basis symbols of $\mathfrak{I}$:

$$\vdash_K^{\mathfrak{I}} A \Leftrightarrow \models_K^{\bar{\mathfrak{I}}} A \quad .$$

With this result we can now relate logical program definitions to formalizations of partial correctness.

---

*/ A non-universal formula, e.g. $\exists x A(x)$, may be true in $\mathcal{C}$ by virtue of some element of $|\mathcal{C}|$ that is not in $|\bar{\mathcal{C}}|$.

The relationship between logical program definitions and formalizations of partial correctness.

Partial correctness

For a given relation ψ between inputs and outputs, a program is said to be partially correct with respect to (w.r.t.) ψ if, for all terminating computations, the inputs and corresponding outputs satisfy ψ.

More formally, for program $P_{\mathfrak{I}}$ as before, with input $\xi \in |\mathfrak{I}|^n$ and relation ψ on $|\mathfrak{I}|^{n+m}$:

$P_{\mathfrak{I}}(\xi)$ is partially correct w.r.t. ψ if and only if

$$\forall \zeta \in |\mathfrak{I}|^m : [A_{P_{\mathfrak{I}}}(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)] \ .$$

Let $U_P(k,\Theta)$ be a (second-order) formula with $n+m$ -ary predicate symbol Θ and constants $(k_1 \ldots k_n) = k$ being the only free */ symbols not included in the formal basis of $\mathfrak{I}$. $U_P(k,\Theta)$ is said to formalize partial correctness of $P_{\mathfrak{I}}$ if for all relations ψ on $|\mathfrak{I}|^{n+m}$ and all inputs $\xi \in |\mathfrak{I}|^n$:

$U_P(k,\Theta)$ true in $\mathfrak{I}^{\xi\psi}_{k\Theta} \Leftrightarrow P_{\mathfrak{I}}(\xi)$ is partially correct w.r.t. ψ , **/

i.e., $\forall \xi \in |\mathfrak{I}|^n$, $\forall \psi$ on $|\mathfrak{I}|^{n+m}$

$U_P(k,\Theta)$ true in $\mathfrak{I}^{\xi\psi}_{k\Theta} \Leftrightarrow \forall \zeta \in |\mathfrak{I}|^m : [A_{P_{\mathfrak{I}}}(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)]$.

---

*/ As a second-order formula, $U_P(k,\Theta)$ may contain bound occurrences of predicate symbols not included in the formal basis of $\mathfrak{I}$.

**/ Note that U_P must formalize the partial correctness of P without introducing any new sorts of data such as 'stacks' or 'states'; only structure $\mathfrak{I}$ is used.

It has been shown by Manna [10] that such a formula U_P can be used to formalize all the regularly observed properties of programs: correctness, termination, equivalence, etc.

During the computation of $P_{\mathcal{J}}(\xi)$, the only elements of $|\mathcal{J}|$ that can be calculated, and thus affect the computation, are those corresponding to terms constructed from the basis symbols of $\mathcal{J}$ and k. Therefore, if we consider the substructure $\overline{\mathcal{J}_k^{\xi}}$: the computation of P on substructure $\overline{\mathcal{J}_k^{\xi}}$ is identical to the computation on $\mathcal{J}$, and therefore $P_{\overline{\mathcal{J}_k^{\xi}}}(\xi) = P_{\mathcal{J}}(\xi)$.

Hence if $\bar{\psi}$ represents the relation ψ restricted to $|\overline{\mathcal{J}_k^{\xi}}|^{n+m}$, then

$P_{\mathcal{J}}(\xi)$ is partially correct w.r.t. ψ

$\Leftrightarrow$ $P_{\overline{\mathcal{J}_k^{\xi}}}(\xi)$ is partially correct w.r.t. $\bar{\psi}$.

$\therefore$ An equivalent condition for $U_P(\xi,\phi)$ to be a formalization of partial correctness of P is:

$\forall \xi \in |\mathcal{J}|$ and $\forall \psi$ on $|\overline{\mathcal{J}_k^{\xi}}|^{n+m}$

$U_P(k,\Theta)$ true in $\overline{\mathcal{J}_{k\Theta}^{\xi}}\,\psi \Leftrightarrow \forall \zeta \in |\overline{\mathcal{J}_k^{\xi}}|^m : [A_{P_{\mathcal{J}}}(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)]$.

Logical program definition

Let (first-order) formula $W_P(k,\phi)$ be an adequate logical definition of P ; then $W_P(k,\phi)$ satisfies condition A) which we repeat below.

A) $\forall \xi \in |\mathfrak{I}|^n$, $\forall \zeta \in |\mathfrak{I}|^m$:

$$A_{P_{\mathfrak{I}}}(\xi,\zeta) \Leftrightarrow \exists \tau = (\tau_1,\ldots,\tau_m)$$
$$\text{s.t.} \quad \mathfrak{I}(\tau) = \zeta \quad \text{and} \quad \vdash^{\mathfrak{I}^{\xi}_k}_{W_P(k,\phi)} \phi(k,\tau) \quad .$$

Since $W_P(k,\phi)$ satisfies A) if and only if the closure of $W_P(k,\phi)$ satisfies A), we can assume in the rest of this section that $W_P(k,\phi)$ is closed. As for partial correctness, $A_{P_{\mathfrak{I}}}(\xi,\zeta)$ can only be true for $\zeta \in |\overline{\mathfrak{I}^{\xi}_k}|^m$, and $\mathfrak{I}[\tau] = \zeta$ for constant terms τ implies $\zeta \in |\overline{\mathfrak{I}^{\xi}_k}|^m$. Therefore, we can equivalently restrict ζ to $|\overline{\mathfrak{I}^{\xi}_k}|^m$. Also if the logical system includes equality we can give an equivalent but more concise condition.

A') $\forall \xi \in |\mathfrak{I}|^n$, $\forall \zeta \in |\overline{\mathfrak{I}^{\xi}_k}|^m$:

$$A_{P_{\mathfrak{I}}}(\xi,\zeta) \Leftrightarrow \vdash^{\mathfrak{I}^{\xi\zeta}_{kh}}_{W_P(k,\phi)} \phi(k,h) \quad .^{*/}$$

---

*/ $h = (h_1 \ldots h_m)$ are constant symbols not in the formal basis of $\mathfrak{I}$.

If $W_P(k,\phi)$ is a universal formula, with only predicate symbols ϕ and $q = (q_1,\ldots,q_j)$ that are not in the formal basis of $\mathcal{I}$, by the $\mathcal{I}$-relative completeness theorem, A') is equivalent to

$$\forall \xi \in |\mathcal{I}|^n \quad , \quad \forall \zeta \in |\overline{\mathcal{I}_k^{\xi}}|^m$$

$$\models_{W_P(k,\phi)} \overline{\mathcal{I}_{k\,h}^{\xi\,\zeta}} \phi(k,h) \quad \Leftrightarrow \quad A_{P_{\mathcal{I}}}(\xi,\zeta) \qquad */$$

i.e., $\forall \xi \in |\mathcal{I}|^n \quad , \quad \forall \zeta \in |\overline{\mathcal{I}_k^{\xi}}|^m$

$$\forall \phi [\exists q W_P(k,\phi) \supset \phi(k,h)] \text{ true in } \overline{\mathcal{I}_{k\,h}^{\xi\,\zeta}} \qquad (1)$$

$$\Leftrightarrow \quad A_{P_{\mathcal{I}}}(\xi,\zeta) \quad .$$

Note that W_P must define P without introducing any new sorts of data such as 'stacks' or 'states'; only structure $\overline{\mathcal{I}_k^{\xi}}$ is used. Now (1) is equivalent to

$$\forall \xi \in |\mathcal{I}|^n \quad , \quad \forall \psi \text{ over } |\overline{\mathcal{I}_k^{\xi}}|^{n+m} \quad .$$

$$\forall \zeta \in |\overline{\mathcal{I}_k^{\xi}}|^m : [A_{P_{\mathcal{I}}}(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)]$$

$$\Leftrightarrow \forall x [\forall \phi [\exists q W_P(k,\phi) \supset \phi(k,x)] \supset \theta(k,x)] \text{ true in } \overline{\mathcal{I}_{k\,\theta}^{\xi\,\psi}} \qquad **/$$

Comparing this with the definition of partial correctness, we get:

---

*/ For $\zeta \in |\overline{\mathcal{I}_k^{\xi}}|^m$, $\mathcal{I}_{kh}^{\xi\xi}$ is the same as $\overline{\mathcal{I}_{k\,h}^{\xi\,\zeta}}$, i.e., $\overline{\mathcal{I}_k^{\xi}}$ expanded to include assignment of ζ to h.

**/ $x = (x_1,\ldots,x_m)$.

Correspondence Theorem 1

For first order, universal formula $W_P(k,\phi)$ with extra predicates q not in the formal basis of $\mathfrak{I}$.

$W_P(k,\phi)$ is a logical definition of $P_{\mathfrak{I}}$ if and only if $U_P(k,\Theta)$ is a formalization of partial correctness of $P_{\mathfrak{I}}$, where $U_P(k,\Theta)$ is $\forall x[\forall\phi[\exists q W_P(k,\phi) \supset \phi(k,x)] \supset \psi(k,x)]$.

In practice $W_P(k,\phi)$ may have logical properties which give a neater formulation of the correspondence theorem. In particular, we can define conditions for 'monotonicity' and 'continuity' of such formulae in a way similar to Park [15].

i) $W_P(k,\phi)$ is said to be monotone if, for all structures $\mathfrak{I}$ and all relations σ,ψ on $|\mathfrak{I}|^{n+m}$ and all $\xi\in|\mathfrak{I}|^n$,

$$\forall\zeta\in|\mathfrak{I}|^m : [\sigma(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)]$$

implies

$$\exists q W_P(k,\phi) \text{ true in } \mathfrak{I}^{\xi\sigma}_{k\phi} \Rightarrow \exists q W_P(k,\phi) \text{ true in } \mathfrak{I}^{\xi\psi}_{k\phi}.$$

ii) $W_P(k,\phi)$ is said to be 'quasi-decreasing' (after Tarski [19]), if for all structures $\mathfrak{I}$ and $\forall\xi\in|\mathfrak{I}|^n$ and all sets $\hat{\psi}$ of relations over $|\mathfrak{I}|^{n+m}$

$$\exists q W_P(k,\phi) \text{ true in } \mathfrak{I}^{\xi\psi}_{k\phi} \text{ for all } \psi\in\hat{\psi}$$

$$\Rightarrow \exists q W_P(k,\phi) \text{ true in } \mathfrak{I}^{\xi\cap\hat{\psi}}_{k\phi}$$

where

$$\forall \xi, \zeta \in |\mathfrak{A}|^n$$

$$\cap\hat{\Psi}(\xi,\zeta) \Leftrightarrow \forall \psi \in \hat{\Psi} : \psi(\xi,\zeta)$$

In a similar way, definitions can be given for 'quasi-increasing', but since this is implied by monotonicity, we need not consider it. However, defining 'continuous' as 'quasi-increasing and quasi-decreasing' (as in Tarski), if $W_P(k,\emptyset)$ satisfies i) - ii) it is monotone and continuous.

Lemma: If $W_P(k,\emptyset)$ is monotone and continuous, for all structures $\mathfrak{A}$ */ and $\forall \xi \in |\mathfrak{A}|^n$, $\forall \psi$ on $|\mathfrak{A}|^{n+m}$:

$$\forall x[\forall \emptyset[\exists q W_P(k,\emptyset) \supset \emptyset(k,x)] \supset \Theta(k,x)] \text{ true in } \mathfrak{A}^{\xi\psi}_{k\Theta}$$

$$\Leftrightarrow \exists q W_P(k,\emptyset) \text{ true in } \mathfrak{A}^{\xi\psi}_{k\emptyset} .$$

Proof

=> L.H.S. equivalent to

$$\forall x[\neg\Theta(k,x) \supset \exists \emptyset[\exists q W_P(k,\emptyset) \wedge \neg \emptyset(k,x)]] \text{ true in } \mathfrak{A}^{\xi\psi}_{k\Theta} .$$

i.e., $\forall \zeta \in |\mathfrak{A}|^m : [\neg \psi(\xi,\zeta) \Rightarrow \exists \eta_\zeta$ on $|\mathfrak{A}|^{n+m} : [\exists q W_P(k,\emptyset)$ true in $\mathfrak{A}_k^{\xi\,\eta_\zeta}_{\emptyset}$ and $\neg \eta_\zeta(\xi,\zeta)]]$.

*/ For the symbols of $W_P(k,\emptyset)$, apart from q , k and $\emptyset$.

$\therefore$ Let $\hat{\eta}$ be the set of relations $\{\eta_\zeta\}$ whose existence is guaranteed by the above formula for each $\zeta \in |\mathfrak{A}|^m$ for which $\neg\, \psi(\xi,\zeta)$

Hence:

i) For each member η_ζ of $\hat{\eta}$, $\exists q W_P(k,\phi)$ is true in $\mathfrak{A}_{k\phi}^{\xi\eta_\zeta}$.

ii) $\cap\hat{\eta}$ has the property

$$\forall \zeta \in |\mathfrak{A}|^n : [\cap\hat{\eta}(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)]$$

because this is equivalent to

$$\forall \zeta \in |\mathfrak{A}|^m : [\forall \eta \in \hat{\eta} : \eta(\xi,\zeta) \Rightarrow \psi(\xi,\zeta)] .$$

This is trivially true for all ζ s.t. $\psi(\xi,\zeta)$, and for all ζ s.t. $\neg\, \psi(\xi,\zeta)$, it is true because $\neg\, \eta_\zeta(\xi,\zeta)$ and $\eta_\zeta \in \hat{\eta}$.

From continuity (quasi-increasing) and i):

$$\exists q W_P(k,\phi) \text{ true in } \mathfrak{A}_{k\phi}^{\xi\cap\hat{\eta}} .$$

Thus, by monotonicity and ii)

$$\exists q W_P(k,\phi) \text{ true in } \mathfrak{A}_{k\phi}^{\xi\psi} \text{, i.e., R.H.S.}$$

$\Leftarrow$ Trivial since for $\forall \xi \in |\mathcal{I}|^n$, $\forall \zeta \in |\mathcal{I}|^m$ and all ψ on $|\mathcal{I}|^{n+m}$:

$\exists q W_P(k,\phi)$ true in $\mathcal{I}^{\xi\psi}_{k\phi}$ and $\forall \phi[\exists q W_P(k,\phi) \supset \phi(k,h)]$ true in $\mathcal{I}^{\xi\zeta}_{kh}$

clearly imply $\psi(\xi,\zeta)$.

Q.E.D.

Noticing that any formula formalizing partial correctness must be both monotone and continuous, from the above result we immediately get a stronger version of the correspondence theorem:

Correspondence Theorem 2.

For first-order universal formula $W_P(k,\phi)$ with extra predicates ϕ and q , not in the formal basis of $\mathcal{I}$:

$W_P(k,\phi)$ is logical definition of $P_{\mathcal{I}}$ and is monotone and continuous if and only if $\exists q W_P(k,\phi)$ is a formalization of partial correctness of $P_{\mathcal{I}}$.

It is interesting that to date all formalizations of partial correctness for programs in various languages have been of the form $\exists q W_P(k,\phi)$. Thus they are all logical definitions of the programs in question. In fact, it is reasonable to claim that the reasoning behind the construction of such formulae was to reflect the execution of the programs.

It also happens that current logical definitions reflect not just the results of computations but the computations themselves. They are

therefore even more closely related to their programs than is strictly 'adequate' as can be seen by considering adequate definitions of two equivalent programs. According to the previous input-output-orientated definition of 'adequate', a logical definition of one is 'adequate' as a logical definition of the other. (This also is true for formalizations of partial correctness.) Therefore logical definitions which describe the computations of programs, are more than just logical definitions of the programs: they describe the 'inner workings' of the programs, the execution of the various pieces from which programs are constructed. Such definitions will be seen to follow from 'adequate' logical definitions of the semantics of programming languages, which will be considered in the next section.

Section 3: LOGICAL LANGUAGE DEFINITION

Logical definition of programming languages

We can say that the purpose of a semantic definition of a programming language is to specify the meaning of programs written in that language. Therefore if Δ_L is a logical definition of language L, then given any program P in L, Δ_L should specify its definition $W_P(k,\phi)$ in some way. The simplest way of doing this for arbitrary program P is to have Δ_L specify a definition $W_P(k,\phi)$ that is related to the computations of P [*] in the following way.

For any definition $W_P(k,\phi)$, the formula $\phi(k,\tau)$ is (ϑ_k^ξ-relatively) deduced from $W_P(k,\phi)$ exactly when the computation of $P(\xi)$ terminates with a result $\vartheta_k^\xi[\tau]$. We can consider formula $\phi(k,\tau)$ as describing a 'situation' in the computation of $P(\xi)$, namely the situation at the end of the computation. Generalizing this, we can imagine other formulae describing intermediate situations in the computation. If these formulae are deduced as intermediate steps in the deduction of $\phi(k,\tau)$, and no others[**], then the deduction is said to describe the computation. By suitable choice of what constitutes a 'situation' it is possible to specify $W_P(k,\phi)$ for any P, i.e., give a definition of the language L. All that is required is that the operation of each basic construct in the program P corresponds to

---

[*] This is not the only solution: Δ_L could find a simpler program equivalent to P, and then produce the definition of this program. This is the practical method used for formalizing the partial correctness of parallel programs in Ashcroft and Manna [2].

[**] i.e., if a formula is deduced of the type that describes situations, then the corresponding situation must occur in the computation.

going from one situation to another. $W_P(k,\phi)$ need only describe the changes in situations produced by the operations of these constructs; formulae describing successive situations will then be deducible from formulae describing earlier situations, and eventually the final situation description $\phi(k,\tau)$ will be deduced.

(All this presupposes that the infinite number of possible situations can be adequately described by logical formulae. We avoid equating 'situations' with 'states' simply because in complicated languages, the 'state' may contain an arbitrary amount of information and may be difficult to describe with a single formula without introducing new sorts of data such as stacks. A 'situation' will in general concern itself with some aspect of the current state, perhaps relating it to previous situations.*/ This will become clear in later sections where definitions of such complicated languages are given.)

We therefore give the following criteria for an 'adequate' logical language definition Δ_L ,

B) <u>The logical language definition conditions</u>

i) Δ_L must specify a logical definition $W_P(k,\phi)$ for any program P .

ii) Each $W_P(k,\phi)$ produced must describe the computations of P , i.e.,

i) there are certain types of formulae that can be interpreted as describing situations in a computation, and

*/ For example, in a multi-level language, it is possible to take a 'situation' as being that part of the state directly affecting the computation at the current level. The inaccessible information at higher levels (e.g. the values of variables temporarily out of scope) will be contained in already deduced formulae describing previous situations. These formulae can be drawn upon later when it is necessary to describe situations when the computation has returned to these higher levels.

ii) from $W_P(k,\emptyset)$ we can (ϑ_k-relatively) deduce exactly these formulae describing the situations that occur in computation of $P_\vartheta(\xi)$.

The way such a definition is realized may vary. For example, Burstall [3] takes Δ_L as being a set of axioms, and also expresses P as axioms, in such a way that the formulae, describing the situations in computations of P , are deducible from the axioms. Clearly, Δ_L is a language definition according to the above criteria. In this work we are going to take Δ_L as an algorithm mapping constructs in a program into formulae describing the effects of these constructs on the general situations that can occur in computations of the program. If we also include a formula specifying the initial situation, the actual situations occurring in the computation will be deducible from these formulae.

We shall illustrate these ideas of program definitions which describe computations using a simple one-level language and show how such a definition $W_P(k,\emptyset)$ relates to a 'Floyd' definition using complete and consistent verification conditions. In the process, we shall develop sufficient conditions on language definitions that are intuitively verifiable. In succeeding sections these conditions will be extended, and definitions given of more complicated languages.

One-level-language definition

A one-level language is one in which the computations of the various parts ('statements') of the program are disjoint (no statement contains

another statement). Hence any computation of a program P simply consists of a concatenation of subcomputations of statements. In this simple language it is possible and natural to take a situation as being the whole state at the end of one subcomputation and the beginning of the next, i.e., when execution is at a point in the program 'between statements'. A description of such a situation consists of the particular point in the program, and the corresponding values of the program variables. This description is achieved by a formula $\phi_e(\tau_1 \ldots \tau_n)$. ϕ_e identifies the point e in the program, and $\tau_1 \ldots \tau_n$ denote the values of the variables at this point.

The successor situation of a given situation is easily described in terms of the effect of executing the next statement.

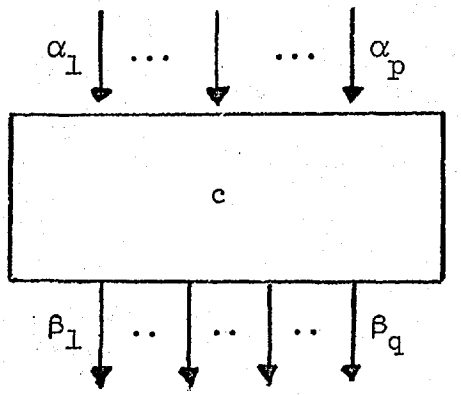
We shall show how a logical definition $W_p(k,\phi)$ of program P can be made up from such descriptions of the effects on situations of the various statements in P .

Let program P with n variables consist of a flowchart constructed from a set C_P of statements, each statement having a certain number of entrances and exits, together with one START and one HALT statement. For simplicity we will assume that a join-point is considered as a type of statement, so that every edge in the program is the exit of one statement and the entrance to another. We associate with each edge e a unique n-ary predicate symbol ϕ_e , associating the special predicate symbols ϕ_o and ϕ with the edges leading from the START statement and to the HALT statement respectively.

The intention is that $\phi_e(\tau_1 \ldots \tau_n)$ is $\mathfrak{I}_k^{\xi}$-derivable from $W_P(k,\phi)$ exactly when the computation of P for input $\xi \in |\mathfrak{I}|^n$ reaches edge e with values $\mathfrak{I}_k^{\xi}[\tau_1], \ldots, \mathfrak{I}_k^{\xi}[\tau_n]$ for the program variables. We will show that $W_P(k,\phi)$ has this property if it consists of a set $\{\phi_o(k)\} \cup \{W_c | c \in C_P\}$ of axioms, where each formula W_c is related to the execution of statement c in a way to be explained below. First we describe execution of c .

For any statement c with p entrances and q exits as shown, $A_{c_{\mathfrak{I}}}$, the graph of c is a $p \times q$ matrix of 2n-ary relations $A_{c\,\mathfrak{I}}^{ij}$, $1 \leq i \leq p$, $1 \leq j \leq q$, such that $\forall \xi \in |\mathfrak{I}|^n$, $\forall \zeta \in |\mathfrak{I}|^n$

$A_{c\,\mathfrak{I}}^{ij}(\xi,\zeta)$ $\Leftrightarrow$ any computation entering c at α_i with variables values ξ leaves c by exit β_j with variable values ζ .



If, for all $1 \leq i \leq p$ we associate a relation ψ_{α_i} on $|\mathfrak{I}|^n$ with entrance α_i, called an input condition for α_i, then for all $1 \leq j \leq q$ we define the relation ψ_{c_j} on $|\mathfrak{I}|^n$ by

$\forall \zeta \in |\mathcal{S}|^n$:

$$\psi_{c_j}(\zeta) \Leftrightarrow \exists \xi \in |\mathcal{S}|^n : [\psi_{\alpha_1}(\xi) \text{ and } A_{c_{\mathcal{S}}}^{1j}(\xi,\zeta)]$$

$$\text{or } [\psi_{\alpha_2}(\xi) \text{ and } A_{c_{\mathcal{S}}}^{2j}(\xi,\zeta)]$$

$$\vdots$$

$$\text{or } [\psi_{\alpha_p}(\xi) \text{ and } A_{c_{\mathcal{S}}}^{pj}(\xi,\zeta)] \ .$$

If we put $\psi_\alpha = (\psi_{\alpha_1} \ldots \psi_{\alpha_p})$, each ψ_{c_j} is called the output condition for β_j corresponding to ψ_α . They have the property that for all inputs to c where the variable values satisfy ψ_α , the variable values on output precisely satisfy the output conditions $\psi_c = (\psi_{c_1} \ldots \psi_{c_q})$. We shall usually denote ψ_c by $\psi_\alpha \cdot A_{c_{\mathcal{S}}}$, because of the similarity to matrix multiplication. (This is the reason $A_{c_{\mathcal{S}}}$ is considered as a matrix.)

We can now give conditions on W_c which ensure that $W_P(k,\emptyset)$ describes the computations of P . We first give rather restrictive conditions which we later relax.

Definition of P (restricted)

$$W_P(k,\emptyset) = \{\emptyset_e(k)\} \cup \{W_c \mid c \in C_P\}$$

where, for c as before:

I) W_c is a first order formula containing predicate symbols $q_c (= q_{c_1} \dots q_{c_j})$ [*], $\phi_{\alpha_1} \dots \phi_{\alpha_p}, \phi_{\beta_1} \dots \phi_{\beta_q}$, and no other symbols not in the basis of $\mathfrak{A}$.

We therefore denote W_c by $W_c(\phi_\alpha, \phi_\beta)$ where

$$\phi_\alpha = (\phi_{\alpha_1} \dots \phi_{\alpha_p}) \quad \text{and} \quad \phi_\beta = (\phi_{\beta_1} \dots \phi_{\beta_q}) .$$

II) For $1 \leq i \leq p$, $1 \leq j \leq q$ and $\zeta, \eta \in |\mathfrak{A}|^n$,

$$A^{ij}_{c\,\mathfrak{A}}(\eta, \zeta) \Rightarrow \vdash^{\mathfrak{A}^{\eta\zeta}_{hg}}_{\phi_{\alpha_i}(h), W_c(\phi_\alpha, \phi_\beta)} \phi_{\beta_j}(g) \quad .$$ [**]

III) For relations ψ_{α_i} on $|\mathfrak{A}|^n$, $i \leq p$, and $\psi_\alpha = (\psi_{\alpha_1} \dots \psi_{\alpha_p})$

$$\exists q_c \tilde{W}_c(\phi_\alpha, \phi_\beta) \text{ is true in } \mathfrak{A}^{\psi_\alpha \psi_\alpha \cdot A_c}_{\phi_\alpha \; \phi_\beta} .$$ [***]

---

[*] The predicate symbols q_c are unique to W_c.

[**] Constants $h = (h_1 \dots h_n)$ and $g = (g_1 \dots g_n)$ are not in the basis of $\mathfrak{A}$.

[***] For any first order formula A, $\tilde{A}$ denotes the closure of A.

That is, there exist relations for q_c such that 'reading' $W_c(\phi_\alpha, \phi_\beta)$ makes sense if we precede it by 'for all sets of computations of c' and read $\phi_{\alpha_i}(\tau_1 \ldots \tau_n)$ as 'one computation enters c by α_i with values $\tau_1 \ldots \tau_n$' and read $\phi_{\beta_j}(\tau_1 \ldots \tau_n)$ as 'one computation leaves c by β_j with values $\tau_1 \ldots \tau_n$'.

In order to show that from $W_p(k, \phi)$ we can ($\mathfrak{I}_k^\xi$-relatively) deduce exactly the successive situations in the computation of $P(\xi)$, it is necessary to indicate just which situations actually occur in $P(\xi)$. This is the purpose of the 'minimal' relations $\hat{\mu}^\xi$ on $|\mathfrak{I}|^n$. Each μ_e^ξ in $\hat{\mu}^\xi$ is defined to be true for exactly those n-tuples of variable values for which the computation $P(\xi)$ reaches edge e. (For the initial and final edges, the relations are μ_o^ξ and μ^ξ respectively.) These relations $\hat{\mu}^\xi$ clearly have the important property that for any statement c in P as before:

for $\mu_\alpha^\xi = (\mu_{\alpha_1}^\xi \ldots \mu_{\alpha_p}^\xi)$ and $\mu_\beta^\xi = (\mu_{\beta_1}^\xi \ldots \mu_{\beta_q}^\xi)$:

$$\mu_\beta^\xi = \mu_\alpha^\xi \cdot A_{c_\mathfrak{I}} \quad .$$

We can now prove the following proposition:

Proposition 1. For

$$W_P(k,\phi) = \{\phi_o(k)\} \cup \{W_c(\phi_\alpha,\phi_\beta) \mid c \in C_P\}$$

where each W_c satisfies the conditions I), II), and III).

$$\forall \xi \in |\mathfrak{I}|^n \quad , \quad \forall \zeta \in |\overline{\mathfrak{I}_k^\xi}|^n :$$

$$\mathfrak{I}_{kh}^{\xi\zeta} \vdash_{W_p(k,\phi)} \phi_e(h) \Leftrightarrow \mu_e^\xi(\zeta) \quad .$$

Proof

$\Rightarrow$ Since $D'(\mathfrak{I}_{kh}^{\xi\zeta})$ is true in all expansions of $\mathfrak{I}_{kh}^{\xi\zeta}$

$$\text{LHS} \Rightarrow \mathfrak{I}_{kh}^{\xi\zeta} \models_{W_P(k,\phi)} \phi_e(h) \quad ,$$

i.e., $\forall \hat{\phi}_e [\exists \hat{q}_c \tilde{W}_P(k,\phi) \supset \phi_e(h)]$ true in $\mathfrak{I}_{kh}^{\xi\zeta}$

where $\hat{\phi}_e$, $\hat{q}_c$ denote all the extra predicate symbols ϕ_e, q_c in $W_P(k,\phi)$.

Now by condition III), and the property of the minimal predicates mentioned previously:

$\exists q_c \tilde{W}_c(\phi_\alpha,\phi_\beta)$ is true for $\mathfrak{I}_{\phi_\alpha \phi_\beta}^{\mu_\alpha^\xi \mu_\beta^\xi}$.

Also $\mu_o^{\xi}(\xi)$ is clearly true.

$\therefore \exists \hat{q}_c \tilde{W}_P(k,\phi)$ is true in $\mathcal{J}_{k\,h\,\phi_e}^{\xi\,\zeta\,\hat{\mu}^{\xi}}$.

Hence $\phi_e^{\xi}(h)$ is true in $\mathcal{J}_{k\,h\,\phi_e}^{\xi\,\zeta\,\hat{\mu}^{\xi}}$,

i.e., $\mu_e^{\xi}(\zeta)$.

$\Leftarrow$ RHS means that the computation of $P(\xi)$ reaches edge e with variable values ζ . It is clear from condition II) that, starting with $\phi_e(k)$, we can trace each step in the computation with deductions from some $W_c(\phi_\alpha,\phi_\beta)$.

$\therefore$ Eventually $\vdash_{W_P(k,\phi)}^{\mathcal{J}_{kh}^{\xi\zeta}} \phi_e(h)$.

Q.E.D.

We see that $W_P(k,\phi)$ describes the computations of P in the desired way.

An obvious corollary to this proposition is

$$\vdash_{W_P(k,\phi)}^{\mathcal{J}_{kh}^{\xi\zeta}} \phi(h) \Leftrightarrow \mu^{\xi}(\zeta) \quad .$$

Since $\mu^{\xi}(\zeta) \Leftrightarrow A_{P_{\mathcal{J}}}(\xi,\zeta)$, $W_P(k,\phi)$ is almost a logical definition of P . To conform with the previous definition of program definitions, we simply add to $W_P(k,\phi)$ the axiom

$$\phi(x) \supset \phi'(k,x) \quad .$$

Then the resulting formula $W_P'(k,\phi')$ will be a logical definition of P .

Note that by describing situations in terms of elements of $|\mathcal{I}|$ and by not allowing the formulae W_c to introduce new sorts of data to describe the executions of statements, we have obtained a definition which itself does not introduce new sorts of data.

Since the only occurrence of ϕ' in $W'(k,\phi')$ is in the axiom $\phi(x) \supset \phi'(k,x)$, $\tilde{W}'(k,\phi')$ is clearly monotone. Also, from the previous proposition, if $\exists \hat{q}_c \tilde{W}_P(k,\phi)$ is true in some expansion of $\mathcal{I}_k^{\xi}$ to include $\hat{\phi}_e$, then, for all e , the relation assigned to ϕ_e includes μ_e^{ξ} . Also, one such expansion is that which assigns $\hat{\mu}^{\xi}$ to $\hat{\phi}_e$. It follows that $\tilde{W}'_P(k,\phi')$ is quasi-decreasing, and therefore continuous.

Hence, by Correspondence Theorem 2, $\exists \hat{q}_c \exists \hat{\phi}_e \tilde{W}'_P(k,\phi')$ is a formalization of partial correctness of $P_{\mathcal{I}}$, if $\tilde{W}'_P(k,\phi')$ is universal.

A logical definition of this one-level language would simply be a specification of formulae W_c for all possible statements c in the language.

It is possible to relax the first condition on the formulae W_c , namely that restricting the extra symbols to the symbols $\hat{\phi}_e$ and $\hat{q}_c$. However, before we do this we can show how this restricted definition is related to one using (similarly restricted) 'verification conditions'.

Definition by 'verification conditions'

Floyd [4] proposed a method of language definition for simple one-level programs using 'verification conditions' for each statement. For statement c as before, the verification condition V_c is a test on relations $\psi_\alpha = \psi_{\alpha_1} \ldots \psi_{\alpha_p}$ and $\psi_\beta = \psi_{\beta_1} \ldots \psi_{\beta_q}$ (on $|\mathcal{S}|^n$) associated with the entrances and exits of c, i.e., $V_c(\psi_\alpha, \psi_\beta)$ is true or false. V_c is said to be consistent if, for all ψ_α, ψ_β, as above

$$V_c(\psi_\alpha, \psi_\beta) \Rightarrow \forall \zeta \in |\mathcal{S}|^n : [\psi_\alpha \cdot A_{c_{\mathcal{S}}}(\zeta) \Rightarrow \psi_\beta(\zeta)] \quad */$$

(that is, if ψ_α and ψ_β satisfy V_c, there is no input to c satisfying ψ_α for which the output does not satisfy ψ_β).

V_c is said to be complete if, for all ψ_α, ψ_β

$$\neg V_c(\psi_\alpha, \psi_\beta) \Rightarrow \exists \zeta \in |\mathcal{S}|^n : [\psi_\alpha \cdot A_c(\zeta) \text{ and } \neg \psi_\beta(\zeta)] \quad **/$$

(that is, if ψ_α and ψ_β do not satisfy V_c, there is some input to c satisfying ψ_α for which the output does not satisfy ψ_β)

---

*/ Putting $\psi_c = \psi_\alpha \cdot A_{c_{\mathcal{S}}}$, $\psi_c(\zeta) \Rightarrow \psi_\beta(\zeta)$ is an abbreviation of $\forall i$, $1 \leq i \leq q : \psi_{c_i}(\zeta) \Rightarrow \psi_{\beta_i}(\zeta)$.

**/ Putting $\psi_c = \psi_\alpha \cdot A_{c_{\mathcal{S}}}$, $\psi_c(\zeta)$ and $\neg \psi_\beta(\zeta)$ is an abbreviation of $\exists i$, $1 \leq i \leq q$ $\psi_{c_i}(\zeta)$ and $\neg \psi_{\beta_i}(\zeta)$.

i.e., $\forall \zeta \in |\mathcal{I}|^n : [\psi_\alpha \cdot A_{c_{\mathcal{I}}}(\zeta) \Rightarrow \psi_\beta(\zeta)] \Rightarrow V_c(\psi_\alpha, \psi_\beta)$.

A set of complete and consistent verification conditions for program $P_{\mathcal{I}}$ is said to be a semantic definition of $P_{\mathcal{I}}$. If the verification conditions can be expressed as formulae, then we shall show how such formulae are related to the logical definition $W_P(k,\phi)$ we have just considered.

Assume that V_c can be expressed as a first order formula Y_c with just predicate symbols q_c, $\phi_{\alpha_1} \ldots \phi_{\alpha_p}, \phi_{\beta_1} \ldots \phi_{\beta_q}$ */ not in the basis of $\mathcal{I}$. That is, if we denote Y_c by $Y_c(\phi_\alpha, \phi_\beta)$ as usual,

$V_c(\psi_\alpha, \psi_\beta) \Leftrightarrow [\exists q_c \tilde{Y}_c(\phi_\alpha, \phi_\beta)$ true in $\mathcal{I}^{\psi_\alpha \psi_\beta}_{\phi_\alpha \phi_\beta}]$. **/

We will show that the formulae Y_c are closely related to the formulae W_c because consistency and completeness are exactly equivalent to conditions II and III (given previously), plus monotonicity.

The definition of monotonicity for $Y_c(\phi_\alpha, \phi_\beta)$ is:

For all $\mathcal{I}$, q-vectors of relations ψ^1_β and ψ^2_β on $|\mathcal{I}|^n$ and p-vector of relations ψ_α on $|\mathcal{I}|^n$:

$$\forall \zeta \in |\mathcal{I}|^n : [\psi^1_\beta(\zeta) \Rightarrow \psi^2_\beta(\zeta)]$$

<u>implies</u>

$$\exists q_c \tilde{Y}_c(\phi_\alpha, \phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha \psi^1_\beta}_{\phi_\alpha \phi_\beta} \Rightarrow \exists q_c Y_c(\phi_\alpha, \phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha \psi^2_\beta}_{\phi_\alpha \phi_\beta} .$$

---

*/ $q_c = (q_{c_1} \ldots q_{c_j})$ are unique to Y_c .

**/ Note that Y_c must express V_c without introducing any new sorts of data such as 'states' or 'stacks'; only structure $\mathcal{I}$ is used.

Theorem 1. For universal formula Y_c, representing verification condition V_c, Y_c satisfies II and III and is monotone $\Leftrightarrow$ V_c is consistent and complete.

Proof

=>

Condition III is

$$\forall\psi_\alpha \text{ on } |\mathfrak{I}|^n : \exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathfrak{I}^{\psi_\alpha\,\psi_\alpha\cdot A_{c_\mathfrak{I}}}_{\phi_\alpha\,\phi_\beta} .$$

Hence by monotonocity

$$\forall\psi_\alpha,\forall\psi_\beta \text{ on } |\mathfrak{I}|^n,\ \forall\zeta\in|\mathfrak{I}|^n :$$

$$[\psi_\alpha\cdot A_{c_\mathfrak{I}}(\zeta) \Rightarrow \psi_\beta(\zeta)] \Rightarrow [\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathfrak{I}^{\psi_\alpha\,\psi_\beta}_{\phi_\alpha\,\phi_\beta}] ,$$

i.e., V_c is complete.

Condition II is

for $1 \le i \le p$, $1 \le j \le q$; $\zeta,\xi\in|\mathfrak{I}|^n$

$$A^{ij}_{c_\mathfrak{I}}(\xi,\zeta) \Rightarrow \vdash^{\mathfrak{I}^{\xi\,\zeta}_{h\,g}}_{\phi_{\alpha_i}(h),\,Y_c(\phi_\alpha,\phi_\beta)} \phi_{\beta_j}(g) .$$

Since $D'(\mathfrak{I}^{\xi\,\zeta}_{h\,g})$ is true in all expansions of $\mathfrak{I}^{\xi\,\zeta}_{h\,g}$,

$$A^{ij}_{c_\mathfrak{I}}(\xi,\zeta) \Rightarrow \models^{\mathfrak{I}^{\xi\,\zeta}_{h\,g}}_{\phi_{\alpha_i}(h),\,Y_c(\phi_\alpha,\phi_\beta)} \phi_{\beta_j}(g) ,$$

i.e.,

$$A^{ij}_{c_\mathfrak{I}}(\xi,\zeta) \Rightarrow \forall q_c\forall\phi_\alpha\forall\phi_\beta[\phi_{\alpha_i}(h) \wedge Y_c(\phi_\alpha,\phi_\beta) \supset \phi_{\beta_j}(g)] \text{ true in } \mathfrak{I}^{\xi\,\zeta}_{h\,g} ,$$

or equivalently,

$$1 \leq i \leq p\ ,\quad 1 \leq j \leq q\ ,\quad \forall\psi_\alpha, \forall\psi_\beta \text{ on } |\mathcal{I}|^n\ ;\quad \forall\xi,\zeta\in|\mathcal{I}|^n$$

$$A^{ij}_{c_{\mathcal{I}}}(\xi,\zeta) \Rightarrow [\psi_{\alpha_i}(\xi) \text{ and } [\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha\psi_\beta}_{\phi_\alpha\phi_\beta}] \Rightarrow \psi_{\beta_j}(\zeta)]$$

re-arranging,

$$1 \leq i \leq p\ ,\quad 1 \leq j \leq q\ ,\quad \forall\psi_\alpha,\psi_\beta \text{ on } |\mathcal{I}|^n\ ,\quad \forall\zeta\in|\mathcal{I}|^n$$

$$\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha\psi_\beta}_{\phi_\alpha\phi_\beta} \Rightarrow [\exists\xi\in|\mathcal{I}|^n : [\psi_{\alpha_i}(\xi) \text{ and } A^{ij}_{c_{\mathcal{I}}}(\alpha,\zeta)] \Rightarrow \psi_{\beta_j}(\zeta)]\ ,$$

that is,

$$\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha\psi_\beta}_{\phi_\alpha\phi_\beta} \Rightarrow \forall\zeta\in|\mathcal{I}|^n : [\psi_\alpha{\cdot}A_{c_{\mathcal{I}}}(\zeta) \Rightarrow \psi_{\beta_j}(\zeta)]$$

$\therefore$ V_c is consistent.

$\Leftarrow$

If V_c is complete, since $\psi_\alpha{\cdot}A_{c_{\mathcal{I}}}(\zeta) \Rightarrow \psi_\alpha{\cdot}A_{c_{\mathcal{I}}}(\zeta)$

$$\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha\ \psi_\alpha\cdot A_{c_{\mathcal{I}}}}_{\phi_\alpha\quad\ \phi_\beta}$$

i.e., Y_c satisfies III.

Now for all $\psi^1_\beta,\psi^2_\beta$ on $|\mathcal{I}|^n$, and ψ_α on $|\mathcal{I}|^n$

if $\forall\zeta\in|\mathcal{I}|^n : [\psi^1_\beta(\zeta) \Rightarrow \psi^2_\beta(\zeta)]$

and $\exists q_c Y_c(\phi_\alpha,\phi_\beta)$ true in $\mathcal{I}^{\psi_\alpha\psi^1_\beta}_{\phi_\alpha\phi_\beta}$,

then, by consistency,

$$\forall \zeta \in |\mathcal{I}|^n : [\psi_\alpha \cdot A_{c_{\mathcal{I}}}(\zeta) \Rightarrow \psi^1_\beta(\zeta)]$$

i.e., $\forall \zeta \in |\mathcal{I}|^n : [\psi_\alpha \cdot A_{c_{\mathcal{I}}}(\zeta) \Rightarrow \psi^2_\beta(\zeta)]$

then by completeness

$$\exists q_c Y_c(\phi_\alpha, \phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha \psi^2_\beta}_{\phi_\alpha \phi_\beta}$$

$\therefore$ Y_c is monotone.

If V_c is consistent for some structure $\mathcal{I}$, then it must be consistent on all substructures of $\mathcal{I}$ containing the constants. This is because the domain of such substructure $\mathcal{I}'$ must be closed under the functions of $\mathcal{I}$, and so the computations in $\mathcal{I}'$ of c for inputs from $|\mathcal{I}'|$ must be identical to the computations in $\mathcal{I}$ for the same inputs.

Now if V_c is consistent, the definition of $\psi_\alpha \cdot A_{c_{\mathcal{I}}}$ gives:

for $1 \le i \le p$, $1 \le j \le q$, $\forall \psi_\alpha, \psi_\beta$ on $|\mathcal{I}|^n$, $\forall \xi, \zeta \in |\mathcal{I}|^n$

$$\exists q_c Y_c(\phi_\alpha, \phi_\beta) \text{ true in } \mathcal{I}^{\psi_\alpha \psi_\beta}_{\phi_\alpha \phi_\beta} \Rightarrow [\psi_{\alpha_i}(\xi) \text{ and } A^{ij}_{c_{\mathcal{I}}}(\xi, \zeta) \Rightarrow \psi_{\beta_j}(\zeta)] .$$

From the above argument, we can restrict the structure to $\overline{\mathcal{I}^{\xi}_h}$. Then

$$1 \le i \le p, \quad 1 \le j \le q, \quad \forall \xi \in |\mathcal{I}|^n, \quad \forall \zeta \in |\overline{\mathcal{I}^{\xi}_h}|^n, \quad \forall \psi_\alpha, \psi_\beta \text{ on } |\overline{\mathcal{I}^{\xi}_h}|^n$$

$$A^{ij}_{c\,\mathfrak{I}}(\xi,\zeta) \Rightarrow [\psi_{\alpha_i}(\xi) \text{ and } [\exists q_c Y_c(\phi_\alpha,\phi_\beta) \text{ true in } \overline{\mathfrak{I}^{\xi}_{h}}\,{}^{\psi_\alpha\psi_\beta}_{\phi_\alpha\phi_\beta}] \Rightarrow \psi_{\beta_j}(\zeta)] \ ,$$

that is: $\forall \xi \in |\mathfrak{I}|^n \ , \quad \forall \zeta \in |\overline{\mathfrak{I}^{\xi}_{h}}|^n$

$$A^{ij}_{c\,\mathfrak{I}}(\xi,\zeta) \Rightarrow \forall q_c \forall \phi_\alpha \forall \phi_\beta [\phi_{\alpha_i}(h) \wedge Y_c(\phi_\alpha,\phi_\beta) \supset \phi_{\beta_j}(g)] \text{ true in } \overline{\mathfrak{I}^{\xi\,\zeta}_{h\,g}} \ .$$

Then since Y_c is universal, we can use the corollary of the $\mathfrak{I}$-relative completeness theorem:

$\forall \xi \in |\mathfrak{I}|^n \ , \quad \forall \zeta \in |\overline{\mathfrak{I}^{\xi}_{h}}|^n$

$$A^{ij}_{c\,\mathfrak{I}}(\xi,\zeta) \Rightarrow \vdash^{\mathfrak{I}^{\xi\,\zeta}_{h\,g}}_{\phi_{\alpha_i}(h),Y_c(\phi_\alpha,\phi_\beta)} \phi_{\beta_j}(g) \ .$$

Since $A^{ij}_{c\,\mathfrak{I}}(\xi,\zeta)$ is false for all $\zeta \notin |\overline{\mathfrak{I}^{\xi}_{h}}|^n$,

$\forall \xi,\zeta \in |\mathfrak{I}|^n$

$$A^{ij}_{c\,\mathfrak{I}}(\xi,\zeta) \Rightarrow \vdash^{\mathfrak{I}^{\xi\,\zeta}_{h\,g}}_{\phi_{\alpha_i}(h),Y_c(\phi_\alpha,\phi_\beta)} \phi_{\beta_j}(g)$$

i.e., Y_c satisfies condition II.

Q.E.D.

We therefore see that verification conditions, originally considered by Floyd as being semantic definitions of programs, do in fact give us definitions of programs in our sense. However, we have been quite restrictive in requiring that the verification conditions be expressible

by first-order formulae with no extra symbols except for the predicate symbols ϕ_e and $\hat{q}_c$.

We can loosen this requirement, and still show the relationship to logical definition. But we must first loosen the requirements on the formulae W_c comprising such definitions.

Definition of P - (general)

We will loosen the conditions on the formulae W_c by allowing extra function and constant symbols to appear in them. The purpose of these symbols will be to allow the construction of expressions other than those representing the data objects manipulated by programs. The new sorts of data objects thus introduced will be conceptual entities useful for describing the execution of programs, e.g. stacks, program counters, tables, etc.

There will therefore be some extension of $|\mathcal{A}|$ which includes all the desired new data objects. We can consider the operations and relations of $\mathcal{A}$ to be extended arbitrarily to cover these extra objects. The resulting extended structure we will denote by $\mathcal{B}$. The extra function, predicate and constant symbols (which we will denote by F , Q and B) will be required to correspond to particular operations and relations on $|\mathcal{B}|$ and elements of $|\mathcal{B}|$ respectively. There will therefore be an expansion $\mathcal{B}'$ of $\mathcal{B}$, to include F , Q and B , that assigns the appropriate meanings.*/ (The operations may, for example,

*/ The formal basis symbols of $\mathcal{B}'$ will consist of the basis symbols of $\mathcal{A}$ and F , Q and B .

be 'pushing' and 'popping' operations, or table look-ups. The relations may be tests for empty stacks or tables.) The formula $W_P(k,\phi)$ must restrict the meanings of the symbols F , Q and B sufficiently to essentially specify $\mathcal{J}'$. We assume therefore that $W_P(k,\phi)$ contains a set of axioms $\mathcal{A}$ for this purpose. Then, if

$$W_P(k,\phi) = \{q_o(k),\mathcal{A}\} \cup \{W_c | c \in C_P\} \quad ,$$

the following conditions on the formulae W_c also comprise conditions on $\mathcal{A}$.

Ia) W_c is a first-order formula containing only predicate symbols $q_c, \phi_{\alpha_1} \ldots \phi_{\alpha_p}, \phi_{\beta_1} \ldots \phi_{\beta_q}$ not in the basis of $\mathcal{J}'$. We denote W_c by $W_c(\phi_\alpha, \phi_\beta)$ as before.

IIa) For $1 \leq i \leq p$, $1 \leq j \leq q$, $\forall \xi, \zeta \in |\mathcal{J}|^n$

$$A^{ij}_{c\,\mathcal{J}}(\xi,\zeta) \Rightarrow \vdash^{\mathcal{J}^{\xi\,\zeta}_{h\,g}}_{\phi_{\alpha_i}(h), W_c(\phi_\alpha,\phi_\beta), \mathcal{A}} \phi_{\beta_j}(g) \quad .$$

IIIa) For relations ψ_α , on $|\mathcal{J}|^n$:

$$\exists q_c \tilde{W}_c(\phi_\alpha,\phi_\beta) \text{ true in } \mathcal{J}'^{\psi_\alpha\, \psi_\alpha \cdot A_{c\mathcal{J}}}_{\phi_\alpha\ \ \phi_\beta} \quad .*)$$

That is, with the desired meaning of the extra symbols F , Q and B , there exist relations for q_c such that reading $W_c(\phi_\alpha,\phi_\beta)$ makes sense if we precede it by 'for all sets of computations of c ', and read $\phi_{\alpha_i}(\tau_1 \ldots \tau_n)$ as 'one computation enters c by α_i with values $\tau_1 \ldots \tau_n$ ', and read $\phi_{\beta_j}(\tau'_1 \ldots \tau'_n)$ as 'one computation leaves c by β_j with values $\tau'_1 \ldots \tau'_n$ '.

*) ψ_α is extended to $\mathcal{J}$ by making it false for any arguments not in $|\mathcal{J}|^n$.

Proposition 2. For

$$W_P(k,\phi) = \{\phi_o(k), \mathcal{A}\} \cup \{W_c(\phi_\alpha, \phi_\beta) \mid c \in C_P\} \quad ,$$

where each W_c satisfies Ia), IIa), and IIIa),

$$\forall \xi \in |\mathcal{J}|^n \quad , \quad \forall \zeta \in |\overline{\mathcal{J}_k^\xi}|^n :$$

$$\vdash_{W_P(k,\phi)}^{\mathcal{J}_{k\,h}^{\xi\,\zeta}} \phi_e(h) \Leftrightarrow \mu_e^\xi(\zeta) \quad .$$

Proof

Since $D'(\mathcal{J}_{h\,k}^{\xi\,\zeta})$ is true in all expansions of any extension of $\mathcal{J}_{h\,k}^{\xi\,\zeta}$, specifically $\mathcal{J}_{k\,h}^{\xi\,\zeta}$,

$$\vdash_{W_P(k,\phi)}^{\mathcal{J}_{k\,h}^{\xi\,\zeta}} \phi_e(h) \Rightarrow \models_{W_P(k,\phi)}^{\mathcal{J}_{k\,h}^{\xi\,\zeta}} \phi_e(h)$$

$$\Rightarrow \forall \hat{\phi}_e [\exists \hat{q}_c W_P(k,\phi) \supset \phi_e(h)] \text{ true in all expansions of } \mathcal{J}_{k\,h}^{\xi\,\zeta}$$

$$\Rightarrow \forall \hat{\phi}_e [\exists \hat{q}_c \tilde{W}_P(k,\phi) \supset \phi_e(h)] \text{ true in } \mathcal{J}'^{\xi\,\zeta}_{k\,h} \quad .$$

Using this result, the proof of this proposition is identical to the proof of Proposition 1, with $\mathcal{J}$ replaced by $\mathcal{J}'$.

Q.E.D.

Then, as before, we immediately get that

$$W'_P(k,\phi') = \{q_o(k), \mathcal{A}, \phi(x) \supset \phi'(k,x)\} \cup \{W_c(\phi_\alpha,\phi_\beta) \mid c \in C_P\}$$

is a logical definition of $P_\mathcal{I}$.

Note that since it introduces new sorts of data, there is no way in which $W'_P(k,\phi')$ can be related to a formalization of partial correctness.

In practice, condition IIa) is satisfied by separate conditions on W_c and $\mathcal{A}$, as follows:

IIa i) $\quad 1 \leq i \leq p$, $\quad 1 \leq j \leq q$, $\quad \forall \xi,\zeta \in |\mathcal{I}|^n$

$$A^{ij}_{c\,\mathcal{I}}(\xi,\zeta) \;\Rightarrow\; \vdash^{\mathcal{I}'\,\xi\,\zeta}_{\phi_{\alpha_i}(h),\,W_c(\phi_\alpha,\phi_\beta)} \phi_{\beta_j}(g) \quad .$$

IIa ii) For all formulas A

$$\vdash^{\mathcal{I}'} A \;\Leftrightarrow\; \vdash^{\mathcal{I}}_{\mathcal{A}} A \quad .$$

If we have axioms $\mathcal{A}$ satisfying IIa ii), then we can show how a semantic definition of P using (general) verification conditions is related to the above definition $W_P(k,\phi)$.

Definition by verification conditions - (general)

Given structure $\mathcal{I}'$ for the extra symbols F , Q and B , as above, we can loosen the conditions on the formulae Y_c representing the verification conditions V_c . We allow Y_c to be a first order

formula containing the extra symbols F, Q and B as well as predicate symbols $q_c, \phi_\alpha, \phi_\beta$; and require that $\forall \psi_\alpha, \psi_\beta$ on $|\mathcal{I}|^n$

$$V_c(\psi_\alpha, \psi_\beta) \Leftrightarrow [\exists q_c \tilde{Y}_c(\phi_\alpha, \phi_\beta) \text{ true in } \mathcal{I}'^{\psi_\alpha \psi_\beta}_{\phi_\alpha \phi_\beta}]. \quad */ \qquad (1)$$

Theorem 2. For universal formula Y_c, representing verification condition V_c, and axioms $\mathcal{A}$ satisfying IIa ii):

Y_c satisfies IIa) and IIIa) and is monotone

$\Leftrightarrow$ V_c is consistent and complete.

Proof

Noting that the only difference in the way Y_c represents V_c is the replacement of $\mathcal{I}'$ for $\mathcal{I}$, we can follow the proof of Theorem 1 to get:

Y_c satisfies IV and IIIa) and is monotone

$\Leftrightarrow$ V_c is consistent and complete,

where condition IV is

$$1 \le i \le p, \quad 1 \le j \le q, \quad \forall \xi, \zeta \in |\mathcal{I}|^n$$

$$A^{ij}_{c\mathcal{I}}(\xi, \zeta) \Rightarrow \vdash^{\mathcal{I}'^{\xi\ \zeta}_{h\ g}}_{\phi_{\alpha_i}(h), Y_c(\phi_\alpha, \phi_\beta)} \phi_{\beta_j}(g) \quad .$$

---

*/ ψ_α, ψ_β are extended to $\mathcal{I}$ by making them false for all arguments not in $|\mathcal{I}|^n$.

(Note that IIIa) is like III with $\mathcal{I}$ replaced by $\mathcal{I}'$.) Then from condition IIa ii) on $\mathcal{A}$,

condition IV is identical to condition IIa).

Q.E.D.

We have not allowed completely general second-order formulas when representing verification conditions, but we have been sufficiently general to claim that a definition by verification conditions is usually a logical definition in our sense.

Conclusion

The example of a one-level language has illustrated how program definitions can be built up from subformulae describing the effects of constructs in the program. The essential properties of the subformulae are

i) They can copy the effect of execution of the construct by $\mathcal{I}$-relative deduction.

ii) They make sense for all sets of computations when $\phi_e(\tau_1 \ldots \tau_n)$ is read as 'one computation is at edge e with values $\tau_1 \ldots \tau_n$ '.

These are properties which can be easily verified from intuitive knowledge of the execution of constructs. These principles can be extended to more complicated, multi-level languages, as will be done in later sections. The justification for the 'adequacy' of the formulae produced will be based on intuitive arguments concerning the above two properties.

This example has also shown how Floyd's verification conditions are strongly related to logical definitions of the type we are considering.

We have also seen how the equivalence of logical definition with formalization of partial correctness does not hold for definitions that introduce new sorts of data. Since we can formalize all properties of programs in terms of partial correctness (Manna [10]), it is desirable that this equivalence holds whenever possible. Therefore, the logical definitions given later go to great pains to avoid introducing new sorts of data.

One method of logical definition which does introduce such new sorts of data is that of Burstall [3]. His definition of Algol is 'adequate' since it has properties similar to those above. However, since it introduces extra function symbols, it is not equivalent to a formalization of partial correctness, and new techniques have to be used to prove properties the programs so defined. His method has the great advantage that it describes the execution of programs using concepts familiar in programming. It therefore seems capable of tackling many practical programming languages, and is a good meta-programming language.

In the following two sections we give definitions of a functional language and of a subset of Algol. The logical definitions of programs so produced are also formalizations of partial correctness, and can be used to prove properties such as termination and correctness, where appropriate.

Section 4: FUNCTIONAL PROGRAMS

In this section we are going to give a logical definition of a functional language. This type of language is essentially multi-level and is simple enough to illustrate the definitional techniques for dealing with multi-level computation. In the next section we will combine the one-level and multi-level techniques in a definition of a large subset of Algol.

Syntax of Functional Programs

A functional program $P_{\mathcal{J}}$ consists of a set $\{F_0 \ldots F_N\}$ of 'specified' functions, with F_0 as the 'initial fnnction'. A function is specified in terms of the other specified functions and basic functions (functions in $\mathcal{J}$). The basic functions we will call constant functions, and the specified functions we will call function variables.

A function variable F_i is 'specified' by an expression (specification) of the form

$$F_i(x_1,\ldots,x_{n_i}) \Leftarrow \tau_i(x_1,\ldots,x_{n_i})$$

or by an expression of the form

$$F_i(x_1,\ldots,x_{n_i}) \Leftarrow \begin{array}{l} \pi_{i_1}(x_1 \ldots x_{n_i}) \rightarrow \tau_{i_1}(x_1 \ldots x_{n_i}), \\ \quad\vdots \qquad\qquad\quad \vdots \\ \pi_{i_{m_i}}(x_1 \ldots x_{n_i}) \rightarrow \tau_{i_{m_i}}(x_1 \ldots x_{n_i}) \end{array}$$

where each $\tau(x_1 \ldots x_{n_i})$ is a term, and each $\pi(x_1,\ldots,x_{n_i})$ is a propositional term:

A term $\tau(x_1,\ldots,x_{n_i})$ is a term in the normal sense constructed from the basis symbols of $\mathcal{L}$ together with the function variables $F_0 \ldots F_N$ and (at most) the variables $x_1,\ldots,x_{n_i}$. We will denote the constants and function symbols in the basis of $\mathcal{L}$ by indexed letter b's, and indexed letter f's respectively. Hence, examples of terms are

$$F_0(f_1(x_1,b_2),b_3)$$

$$f_1(x_1,x_2,x_4,b_1)$$

$$f_2(F_1(F_1(f_1(x_1))),x_2) \quad .$$

A variable-free term is a term without variables. A simple term is a term without function variables. A constant term is a term with neither variables nor function variables.

A propositional term $\pi(x_1,\ldots,x_{n_i})$ is an expression of one of the following forms:

i) $P_j(\tau_1(x_1 \ldots x_{n_i}),\ldots,\tau_{n_j}(x_1 \ldots x_{n_i}))$

where P_j is some predicate symbol in the basis of $\mathcal{L}$, and $\tau_1 \ldots \tau_{n_j}$ are simple terms.

ii) $\neg\, \pi_1(x_1,\ldots,x_{n_i})$

iii) $[\pi_1(x_1 \ldots x_{n_i}) \wedge \pi_2(x_1 \ldots x_{n_i})]$

iv) $[\pi_1(x_1 \ldots x_{n_i}) \vee \pi_2(x_1 \ldots x_{n_i})]$

where π_1,π_2 are propositional terms.

Note that a propositional term is a quantifier-free formula in the basis symbols of $\mathcal{I}$, with free variables. For an assignment of values from $|\mathcal{I}|$ to the variables $x_1,\ldots,x_{n_i}$, such a propositional term is true or false in $\mathcal{I}$. We stipulate that the propositional terms in a specification must be mutually exclusive, i.e., there is no assignment of values to the variables for which two such propositional terms are simultaneously true in $\mathcal{I}$.

Examples of propositional terms are

$$P_1(x_1)$$

$$[P_1(x_1) \wedge \neg P_2(x_2, f_2(x_3, b_1))]$$

$$[\neg[P_1(x_1) \vee P_2(x_2)] \wedge P_4(f_1(x_1,x_2), b_4)] \quad .$$

Example of functional programs are

i) $P_{1_{\mathcal{I}}} = \{F_0\}$

where $F_0(x_1) \Leftarrow P_1(x_1, b_2) \rightarrow b_1$,

$$\neg P_1(x_1, b_2) \rightarrow f_1(x_1, F_0(f_2(x_1, b_1))) \quad .$$

ii) $P_{2_{\mathcal{I}}} = \{F_0, F_1\}$

where $F_0(x_1,x_2) \Leftarrow [\neg P_1(x_1,x_2) \wedge P_2(x_2,x_1)] \rightarrow b_2$,

$$[\neg P_2(x_2,x_1) \wedge P_2(f_1(x_2,b_3),x_1)] \rightarrow F_1(x_1, f_2(x_1,x_2)) \ ,$$

$$\neg P_2(f_1(x_2,b_3),x_1) \rightarrow F_1(x_1,x_2)$$

and $\quad F_1(x_1,x_2) \Leftarrow P_1(x_2,b_2) \rightarrow b_1$,

$$\neg P_1(x_2,b_2) \rightarrow f_3(f_1(x_1,F_1(f_2(x_1,b_1),f_2(x_2,b_1))),x_2) \ .$$

If $|\mathcal{I}|$ consists of the integers, and

b_1 is 1 ; b_2 is 0 ; b_3 is 2 ;

and

$P_1(x_1,x_2)$ means $x_1 = x_2$

$P_2(x_1,x_2)$ means $x_1 \geq x_2$

and

$f_1(x_1,x_2)$ means $x_1 * x_2$ (multiplication)

$f_2(x_1,x_2)$ means $x_1 - x_2$

$f_3(x_1,x_2)$ means x_1 / x_2 (integer division)

then we may re-write $P_{1_{\mathcal{I}}}$ and $P_{2_{\mathcal{I}}}$ in the more usual way:

i) $P_{1_{\mathcal{I}}} = \{F_0\}$

where $\quad F_0(x_1) \Leftarrow x_1 = 0 \rightarrow 1$,

$$x_1 \neq 0 \rightarrow x_1 * F_0(x_1-1)$$

and

ii) $P_{2_{\mathcal{I}}} = \{F_0,F_1\}$

where $\quad F_0\{x_1,x_2\} \Leftarrow x_2 \geq x_1 \wedge x_2 \neq x_1 \rightarrow 0$,

$$2*x_2 \geq x_1 \wedge x_2 \not\geq x_1 \rightarrow F_1(x_1,x_1-x_2) \ ,$$

$$2*x_2 \not\geq x_1 \rightarrow F_1(x_1,x_2)$$

$$F_1(x_1,x_2) \Leftarrow x_2 = 0 \rightarrow 1 \ ,$$

$$x_2 \neq 0 \rightarrow (x_1 * F_1(x_1-1,x_2-1))/x_2 \ .$$

Semantics of Functional Programs

An intuitive description of the semantics of functional programs will be given, which will be used later to justify a logical definition.

a) For inputs $\xi_1,\ldots,\xi_{n_o} \in |\mathfrak{I}|$, the program $P_{\mathfrak{I}}$ 'invokes' or 'calls' the initial function variable F_0 with arguments $\xi_1,\ldots,\xi_{n_o}$. When computation of F_0, called with $\xi_1,\ldots,\xi_{n_o}$, terminates with a value $\zeta \in |\mathfrak{I}|$, this is the result of program $P_{\mathfrak{I}}$ for input $\xi_1,\ldots,\xi_{n_o}$.

b) The computation of a function variable F_i called with arguments $\eta_1,\ldots,\eta_{n_i} \in |\mathfrak{I}|$ is determined by its specification as follows:

i) For specification

$$F_i(x_1,\ldots,x_{n_i}) \Leftarrow \tau_i(x_1,\ldots,x_{n_i})$$

the (variable free) term $\tau_i(\eta_1,\ldots,\eta_{n_i})$ */ is computed.

ii) For specification

$$F_i(x_1,\ldots,x_{n_i}) \Leftarrow \begin{array}{l} \pi_{i_1}(x_1\ldots x_{n_i}) \rightarrow \tau_{i_1}(x_1\ldots x_{n_i}), \\ \quad\vdots \qquad\qquad\qquad\quad \vdots \\ \pi_{i_{m_i}}(x_1\ldots x_{n_i}) \rightarrow \tau_{i_{m_i}}(x_1\ldots x_{n_i}) \end{array}$$

---

*/ For the purpose of describing computations we could add to the symbols of $\mathfrak{I}$ a name i^o for each element $i \in |\mathfrak{I}|$. Then strictly we would say that $\tau_i(\eta_1^o,\ldots,\eta_{n_i}^o)$ is computed. However, we feel no confusion will arise if we refer to $\tau_i(\eta_1\ldots\eta_{n_i})$.

there will be at most one propositional term π_{i_j} true in $\mathcal{I}$ for $\eta_1,\ldots,\eta_{n_i}$ assigned to the variables. If there is no such propositional term, then the computation of F_i, called with $\eta_1\ldots\eta_{n_i}$, is suspended. (A suspended computation never terminates with a value.) Otherwise, if propositional term $\pi_{i_j}(\eta_1,\ldots,\eta_{n_j})$ is true in $\mathcal{I}$, then (variable-free) term $\tau_{i_j}(\eta_1,\ldots,\eta_{n_i})$ is computed.

In both cases i) and ii), if the computation of the term terminates with a value, then the computation of F_i called with $\eta_1\ldots\eta_{n_i}$ terminates with this value.

c) A variable free term τ is computed as follows. If τ is a constant term, then computation of τ terminates immediately with the value $\mathcal{I}[\tau]$.

Otherwise, τ must contain one or more subterms of the form $F_j(\tau_1,\ldots,\tau_{n_j})$, where $\tau_1\ldots\tau_{n_j}$ are constant terms (i.e., these are the smallest non-simple subterms of τ). For all such subterms, like $F_j(\tau_1,\ldots,\tau_{n_j})$, we call F_j with arguments $\mathcal{I}[\tau_1],\ldots,\mathcal{I}[\tau_{n_j}]$. When computation of any of these subterms terminates with a value, we replace the subterm in τ by this value. As soon as any new non-simple subterms are produced, they are computed in the same manner, and the process continues until all non-simple subterms have been removed, and the computation terminates with a value.

For example, computation of term

$$F_0(F_0(\eta_1,f_2(\eta_2)),F_1(f_1(F_1(f_1(\eta_1,b_2)),F_2(\eta_2))))$$

proceeds as follows:

I) F_0 is called with $\eta_1, \mathcal{S}[f_2(\eta_2)]$ and

F_1 is called with $\mathcal{S}[f_1(\eta_1, b_2)]$ and

F_2 is called with η_2 .

II) When F_1 , called with $\mathcal{S}[f_1(\eta_1, b_2)]$, terminates with ζ_1

and F_2 , called with η_2 , terminates with ζ_2

F_1 is called with $\mathcal{S}[f_2(\zeta_1, \zeta_2)]$.

III) When F_1 , called with $\mathcal{S}[f_2(\zeta_1, \zeta_2)]$, terminates with ζ_3

and F_0 , called with η_1, $\mathcal{S}[f_2(\eta_2)]$ terminates with ζ_4

F_0 is called with ζ_4, ζ_3 .

IV) When F_0 , called with ζ_4, ζ_3 , terminates with ζ_5

computation of τ terminates with ζ_5 .

This completes the description of the computation of functional programs. We call such programs 'multi-level' because the computations of parts of the program (function variables) contain the computations of other parts of the program (other function variables).

There are clearly two ways in which the computation of some term could fail to terminate. Either the computation proceeds forever, calling ever more function variables, or else some function variable is called with arguments for which its computation is suspended. The situations are intrinsically different, in that the latter can be detected during computation; it is intended to be an error condition.

The description of computation clearly conforms with the usual intuitive meaning of the execution of functional programs. We will not give examples of computations, but merely remark that for functional programs $P_{1_\mathcal{J}}$ and $P_{2_\mathcal{J}}$ given previously:

i) For any non-negative integer input n, computation of $P_{1_\mathcal{J}}$ terminates with result $n!$

ii) For any non-negative integer inputs n,m, computation of $P_{2_\mathcal{J}}$ terminates with result ${}_nC_m = \frac{n!}{m!(n-m)!}$.

Logical Definition of Functional Programs

With each function variable specification σ we associate an axiom W_σ as illustrated by the following examples. The construction of W_σ should be clear if $q_{F_i}(\tau_1 \ldots \tau_{n_i})$ is read as 'F_i is called with $\tau_1,\ldots,\tau_{n_i}$', and $Q_{F_i}(\tau_1,\ldots,\tau_{n_i},\tau_{n_i+1})$ is read as 'F_i, called with $\tau_1,\ldots,\tau_{n_i}$, terminates with value τ_{n_i+1}', and $E_{F_i}(\tau_1,\ldots,\tau_{n_i})$ is read as 'computation of F_i, called with $\tau_1 \ldots \tau_{n_i}$, is suspended'.

i) σ : $F_1(x_1,x_2,x_3) \Leftarrow f_1(x_1,f_2(x_2,x_3))$

W_σ : $q_{F_1}(x_1,x_2,x_3) \supset Q_{F_2}(x_1,x_2,x_3,f_1(x_1,f_2(x_2,x_3)))$

ii) σ : $F_1(x_1,x_2,x_3) \Leftarrow \pi_{1_1}(x_1,x_2,x_3) \rightarrow f_1(x_1,f_2(x_2,x_3))$,

$\pi_{1_2}(x_1,x_3) \rightarrow F_3(x_1)$

for propositional terms π_{1_1},π_{1_2} .

$$W_\sigma : q_{F_1}(x_1,x_2,x_3) \supset \{[\pi_{1_1}(x_1,x_2,x_3) \supset Q_{F_1}(x_1,x_2,x_3,f_1(x_1,f_2(x_2,x_3)))]$$
$$\wedge\ [\pi_{1_2}(x_1,x_3) \supset \{q_{F_3}(x_1) \wedge [Q_{F_3}(x_1,z_1) \supset$$
$$Q_{F_1}(x_1,x_2,x_3,z_1)]\}]$$
$$\wedge\ [[\neg\, \pi_{1_1}(x_1,x_2,x_3) \wedge \neg\, \pi_{1_2}(x_1,x_3)] \supset E_{F_i}(x_1,x_2,x_3)]\} \quad .$$

iii) σ : $F_1(x_1,x_2,x_3) \Leftarrow \pi_{1_1}(x_1,x_2,x_3) \rightarrow F_3(x_1)$,

$$\pi_{1_2}(x_1,x_3) \rightarrow f_2(F_2(F_3(f_1(x_1,x_2)),x_3))$$

$$W_\sigma : q_{F_1}(x_1,x_2,x_3) \supset \{[\pi_{1_1}(x_1,x_2,x_3) \supset \{q_{F_3}(x_1)$$
$$\wedge\ [Q_{F_3}(x_1,z_1) \supset Q_{F_1}(x_1,x_2,x_3,z_1)]\}]$$
$$\wedge\ [\pi_{1_2}(x_1,x_3) \supset \{q_{F_3}(f_1(x_1,x_2)) \wedge$$
$$[Q_{F_3}(f_1(x_1,x_2),z_2) \supset q_{F_2}(z_2,x_3)] \wedge$$
$$[Q_{F_3}(f_1(x_1,x_2),z_2) \wedge Q_{F_2}(z_2,x_3,z_3)$$
$$\supset Q_{F_1}(x_1,x_2,x_3,f_2(z_3))]\}]$$
$$\wedge\ [[\neg\, \pi_{1_1}(x_1,x_2,x_3) \wedge \neg\, \pi_{1_2}(x_1,x_3)] \supset E_{F_1}(x_1,x_2,x_3)]\} \quad .$$

iv) σ : $F_3(x_1,x_2) \Leftarrow F_0(F_0(x_1,f_2(x_2)),F_1(f_1(F_1(f_1(x_1,b_2)),F_2(x_2))))$.

Note that the term in this example is the one used previously to illustrate computations of terms.

$$W_\sigma : \; q_{F_3}(x_1,x_2) \supset \{q_{F_0}(x_1,f_2(x_2)) \wedge q_{F_1}(f_1(x_1,b_2)) \wedge q_{F_2}(x_2)$$

$$\wedge [[Q_{F_1}(f_1(x_1,b_2),z_1) \wedge Q_{F_2}(x_2,z_2)] \supset q_{F_1}(f_1(z_1,z_2))]$$

$$\wedge [[Q_{F_1}(f_1(x_1,b_2),z_1) \wedge Q_{F_2}(x_2,z_2) \wedge Q_{F_1}(f_1(z_1,z_2),z_3)$$

$$\wedge Q_{F_0}(x_1,f_2(x_3),z_4) \supset q_{F_0}(z_4,z_3)]$$

$$\wedge [[Q_{F_1}(f_1(x_1,b_2),z_1) \wedge Q_{F_2}(x_2,z_2) \wedge Q_{F_1}(f_1(z_1,z_2),z_3)$$

$$\wedge Q_{F_0}(x_1,f_2(x_2),z_4) \wedge Q_{F_0}(z_4,z_3,z_5)] \supset Q_{F_3}(x_1,x_2,z_5)]\} .$$

The method of construction of W_σ for all σ should be apparent from these examples. An algorithm for constructing W_σ could be given, but it would tend to obscure the intuitive meaning behind the construction.

With any program $P_\mathcal{J} = \{F_0 \ldots F_N\}$, where each F_i has specification σ_i, we associate the set of axioms $W_P(k,Q_{F_0})$ [*] defined by

$$W_P(k,Q_{F_0}) = \{q_{F_0}(k_1 \ldots k_{n_o})\} \cup \{W_{\sigma_i} \,|\, F_i \epsilon P_\mathcal{J}\} .$$

(We shall in the future denote $\{W_{\sigma_i} \,|\, F_i \epsilon P_\mathcal{J}\}$ by $\hat{W}_\sigma$.) We prove later that $W_P(k,Q_{F_0})$ is a logical definition of $P_\mathcal{J}$.

---

[*] $k = (k_1, \ldots, k_{n_o})$ are constant symbols.

Examples

i) $W_{P_1}(k_1, Q_{F_0})$ is as follows (the axioms are separated by semicolons):

$$q_{F_0}(k_1);$$

$$q_{F_0}(x_1) \supset \{[P_1(x_1,b_2) \supset Q_{F_0}(x_1,b_1)]$$
$$\wedge [\neg P_1(x_1,b_2) \supset$$
$$\{q_{F_0}(f_2(x_1,b_1)) \wedge$$
$$[Q_{F_0}(f_2(x_1,b_1),z_1) \supset Q_{F_0}(x_1,f_1(x_1,z_1))]\}]$$
$$\wedge [[\neg P_1(x_1,b_2) \wedge P_1(x_1,b_2)] \supset E_{F_0}(x_1)]\};$$

ii) $W_{P_2}(k_1,k_2,Q_{F_0})$ is as follows:

$$q_{F_0}(k_1,k_2);$$

$$q_{F_0}(x_1,x_2) \supset \{[[\neg P_1(x_1,x_2) \wedge P_2(x_2,x_1)] \supset Q_{F_0}(x_1,x_2,b_2)]$$
$$\wedge [[\neg P_2(x_2,x_1) \wedge P_2(f_1(x_2,b_3),x_1)] \supset$$
$$\{q_{F_1}(x_1,f_2(x_1,x_2)) \wedge$$
$$[Q_{F_1}(x_1,f_2(x_1,x_2),z_1) \supset$$
$$Q_{F_0}(x_1,x_2,z_1)]\}]$$
$$\wedge [\neg P_2(f_1(x_2,b_3),x_1) \supset$$
$$\{q_{F_1}(x_1,x_2) \wedge$$
$$[Q_{F_1}(x_1,x_2,z_2) \supset$$
$$Q_{F_0}(x_1,x_2,z_2)]\}]$$
$$\wedge [[\neg[\neg P_1(x_1,x_2) \wedge P_2(x_2,x_1)]$$
$$\wedge \neg[\neg P_2(x_2,x_1) \wedge P_2(f_1(x_2,b_3),x_1)]$$

$$\wedge\ P_2(f_1(x_2,b_3),x_1)] \supset E_{F_0}(x_1,x_2)]\}\ ;$$

$$q_{F_1}(x_1,x_2) \supset \{[P_1(x_2,b_2) \supset Q_{F_1}(x_1,x_2,b_1)]$$

$$\wedge\ [\neg\ P_1(x_2,b_2) \supset$$

$$\{q_{F_1}(f_2(x_1,b_1),f_2(x_2,b_1)) \wedge$$

$$[Q_{F_1}(f_2(x_1,b_1),f_2(x_2,b_1),z_1) \supset$$

$$Q_{F_1}(x_1,x_2,f_3(f_1(x_1,z_1),x_2))]\}]$$

$$\wedge\ [[\neg\ P_1(x_2,b_2) \wedge P_1(x_2,b_2)] \supset E_{F_1}(x_1,x_2)]\}\ ;$$

To prove that $W_P(k,Q_{F_0})$ is a logical definition of P_ϑ, we proceed similarly to the case of the one-level language in the previous section. However, because of the multi-level nature of computations, the conditions that $\hat{W}_\sigma$ must satisfy cannot be stated for each W_{σ_i} separately. Before we can state the conditions we must look more closely at some properties of multi-level computation.

If a function variable F_j occurs in a term in σ_i, and for computation of F_i, called with arguments $\xi_1 \ldots \xi_{n_i}$, this term is computed, causing F_j to be called with arguments $\eta_1 \ldots \eta_{n_j}$, then we say that

'F_j is called with $\eta_1 \ldots \eta_{n_j}$ at the top level of
F_i called with $\xi_1 \ldots \xi_{n_i}$'.

e.g. For

$$F_1(x_1) \Leftarrow F_2(f_1(F_3(f_1(x)), F_2(x)))$$

$$F_3(x_1) \Leftarrow F_2(f_2(x_1))$$

(a) F_2 is called with ξ_1 at the top level of F_1 called with ξ_1;

(b) F_2 is called with $f_2(f_1(\xi_1))$ during the computation of F_1 called with ξ_1, but is <u>not</u> so called at the top level;

(c) If F_2 called with ξ_1 terminates with ζ_1, and
F_2 called with $f_2(f_1(\xi_1))$ terminates with ζ_2,
then F_2 is called with $f_1(\zeta_2, \zeta_1)$ at the top level of
F_1 called with ξ_1.

We can now define a relation $\subset$ on terminating computations of function variables, as follows:

i) $\sigma_i = F_i(x_1 \ldots x_{n_i}) \Leftarrow \tau_i(x_1 \ldots x_{n_i})$

where τ_i is a simple term.

There is no terminating computation X such that

$X \subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}]$, for any $\xi_1 \ldots \xi_{n_i}$.

ii) $\sigma_i = F_i(x_1 \ldots x_{n_i}) \Leftarrow \pi_{i_1}(x_1 \ldots x_{n_i}) \rightarrow \tau_{i_1}(x_1 \ldots x_{n_i})$,

$\vdots$

$\pi_{i_{m_i}}(x_1 \ldots x_{n_i}) \rightarrow \tau_{i_{m_i}}(x_1 \ldots x_{n_i})$

where for arguments $\xi_1 \ldots \xi_{n_i}$, $\pi_{i_j}(\xi_1 \ldots \xi_{n_i})$ is true and $\tau_{i_j}(x_1 \ldots x_{n_i})$ is a simple term.

There is no terminating computation X such that

$X \subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}]$.

iii) In all other cases, if F_{i_j} is called with $\eta_1 \ldots \eta_{n_j}$ at the top level of F_i called with $\xi_1 \ldots \xi_{n_i}$, and F_i called with $\xi_1 \ldots \xi_{n_i}$ terminates, then

$[F_j$ called with $\eta_1 \ldots \eta_{n_j}] \subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}]$.

(F_j called with $\eta_1 \ldots \eta_{n_j}$ clearly must terminate.)

We can extend this relation to suitably defined parts of computations of function variables as follows.

If F_j is called with $\zeta_1 \ldots \zeta_{n_j}$ at the top level of F_i called with $\xi_1 \ldots \xi_{n_i}$, then the computation of F_i called with $\xi_1 \ldots \xi_{n_i}$,

up to the point where F_j is called with $\zeta_1 \ldots \zeta_{n_j}$ at the top level, is called a <u>part</u> of the computation of F_i called with $\xi_1 \ldots \xi_{n_i}$. We will refer to this part as

'F_i called with $\xi_1 \ldots \xi_{n_i}$ up to F_j called with $\zeta_1 \ldots \zeta_{n_j}$'.

Note that all parts of computations (terminating or non-terminating) are finite.

We extend the relation $\subset$ to cover parts of computations by noting that i) and ii) above hold also for parts of computations X. We also add the extra cases:

iv) All parts of F_i called with $\xi_1 \ldots \xi_{n_i} \subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}]$.

v) If F_j is called at the top level of F_i, this is because σ_i contains a term with subterm $F_j(\tau_1 \ldots \tau_{n_j})$. Now for any F_k, if F_k is called with $\eta_1 \ldots \eta_{n_k}$ at the top level of F_i called with $\xi_1 \ldots \xi_{n_i}$, and the occurrence of F_k in question is in one of the terms $\tau_1 \ldots \tau_{n_j}$, then

a) $[F_k$ called with $\eta_1 \ldots \eta_{n_k}]$
$\subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}$ up to F_j called with $\zeta_1 \ldots \zeta_{n_j}]$

b) $[F_i$ called with $\xi_1 \ldots \xi_n$ up to F_k called with $\eta_1 \ldots \eta_k]$
$\subset [F_i$ called with $\xi_1 \ldots \xi_{n_i}$ up to F_j called with $\zeta_1 \ldots \zeta_{n_j}]$.

This completes the definition of the relation $\subset$ on terminating computations and parts of computations.

The transitive closure $\overset{*}{\subset}$ of this relation*/ is the non-reflexive partial ordering which simply reflects the containment of one terminating computation, or part of computation, in another, due to the multi-level nature of computations. We have made this ordering explicit to allow its use in a later inductive argument.

In the rest of this section we use the following convention:

$(\xi_1 \ldots \xi_{n_o}) = \xi \epsilon |\vartheta|^{n_o}$; $(\delta_1 \ldots \delta_{n_i}) = \delta \epsilon |\vartheta|^{n_i}$;

$(\zeta_1 \ldots \zeta_{n_j}) = \zeta \epsilon |\vartheta|^{n_j}$; $(\eta_1 \ldots \eta_{n_k}) = \eta \epsilon |\vartheta|^{n_k}$

$\mu, \gamma \epsilon |\vartheta|$;

$k = (k_1 \ldots k_{n_o})$, $h = (h_1 \ldots h_{n_j})$, $d = (d_1 \ldots d_{n_i})$, $g = (g_1 \ldots g_{n_k})$ and e are constant symbols.

We can now give the conditions that we wish $\hat{W}_\sigma$ to satisfy.

I) If F_j is called with ζ at the top level of F_i called with δ and for all F_k

{[F_i called with δ up to F_k called with η]

$\subset$ [F_i called with δ up to F_j called with ζ]}

$$\Rightarrow \vartheta^{\delta\,\eta}_{d\,g} \vdash^{q_{F_k}(g)}_{q_{F_i}(d),\hat{W}_\sigma}$$

---

*/ relation $\overset{*}{\subset}$ is defined by

i) $X \subset Y \Rightarrow X \overset{*}{\subset} Y$

ii) $X \overset{*}{\subset} Y$ and $Y \overset{*}{\subset} Z \Rightarrow X \overset{*}{\supset} Z$.

and for all F_k :

$$\{[F_k \text{ called with } \eta \text{ (and terminating with } \mu)] \subset [F_i \text{ called with } \delta \text{ up to } F_j \text{ called with } \zeta]\}$$

$$\Rightarrow \vdash^{\vartheta^{\eta\mu}_{g\,e}}_{q_{F_k}(g),\hat{W}_\sigma} Q_{F_k}(g,e)$$

then

$$\vdash^{\vartheta^{\delta\zeta}_{d\,h}}_{q_{F_i}(d),\hat{W}_\sigma} q_{F_j}(h) \quad .$$

II) If F_i called with δ terminates with γ

and for all F_k :

$$\{[F_i \text{ called with } \delta \text{ up to } F_k \text{ called with } \eta] \subset [F_i \text{ called with } \delta]\}$$

$$\Rightarrow \vdash^{\vartheta^{\delta\eta}_{d\,g}}_{q_{F_i}(d),\hat{W}_\sigma} q_{F_k}(g)$$

and for all F_k :

$$\{[F_k \text{ called with } \eta \text{ (and terminating with } \mu)] \subset [F_i \text{ called with } \delta]\}$$

$$\Rightarrow \vdash^{\vartheta^{\eta\mu}_{g\,e}}_{q_{F_k}(g),\hat{W}_\sigma} Q_{F_k}(g,e)$$

then

$$\mathcal{J}^{\delta\,\gamma}_{d\,e} \vdash_{q_{F_i}(d),\hat{W}_\sigma} Q_{F_i}(d,e) \quad .$$

III) For all $W_{\sigma_i} \epsilon \hat{W}_\sigma$, reading W_{σ_i} makes sense if we precede it by 'for all sets of computations of F_i', and $q_{F_j}(\tau_1 \ldots \tau_{n_j})$ is read as 'in one computation, F_j is called with $\tau_1 \ldots \tau_{n_j}$', and $Q_{F_j}(\tau_1 \ldots \tau_{n_j}, \tau_{i_j+1})$ is read as 'in one computation, F_j, called with $\tau_1 \ldots \tau_{n_j}$, terminates with value τ_{n_j+1}', and $E_{F_j}(\tau_1 \ldots \tau_{n_j})$ is read as 'in one computation, computation of F_j, called with $\tau_1 \ldots \tau_{n_j}$, is suspended'.

These conditions have been given in a form in which it is readily seen that $\hat{W}_\sigma$ satisfies them. The required consequences of these conditions can be stated much more concisely than the conditions themselves.

Since there can be no infinitely descending chains of terminating computations ordered by $\overset{*}{\subset}$, we can immediately get more concise forms of I and II by induction (on the partially ordered set).

Ia) <u>If</u> F_j is called with ζ at the top level of F_i called with δ <u>then</u>

$$\mathcal{J}^{\delta\,\zeta}_{d\,h} \vdash_{q_{F_i}(d),\hat{W}_\sigma} q_{F_j}(h) \quad .$$

IIa) If F_i called with δ terminates with γ

then

$$\mathcal{J}^{\delta\,\gamma}_{d\,e} \vdash_{q_{F_i}(d),\hat{W}_\sigma} Q_{F_i}(d,e) \quad .$$

The required consequence of III comes from considering the minimal relations as in the previous section. If for $\xi = (\xi_1 \ldots \xi_{n_o})$ we define the sets of minimal relations $\hat{\mu}^\xi$, $\hat{\nu}^\xi$ and $\hat{\varepsilon}^\xi$ such that

i) for $\mu^\xi_{F_j} \epsilon \hat{\mu}^\xi$, $\mu^\xi_{F_j}(\zeta)$ if and only if F_j is called with ζ in computation of $P_\mathcal{J}$ for inputs ξ.

ii) for $\nu^\xi_{F_j} \epsilon \hat{\nu}^\xi$, $\nu^\xi_{F_j}(\zeta,\gamma)$ if and only if F_j is called with ζ in computation of $P_\mathcal{J}$ for input ξ, and terminates with value γ.

and iii) for $\varepsilon^\xi_{F_j} \epsilon \hat{\varepsilon}^\xi$, $\varepsilon^\xi_{F_j}(\zeta)$ if and only if computation of F_j, called with ζ in computation of $P_\mathcal{J}$ for inputs ξ, is suspended.

By choosing just those computations of F_i that occur in computation of $P_\mathcal{J}$ for inputs ξ, condition III implies that

IIIa) for all $W_{\sigma_i} \epsilon \hat{W}_\sigma$, W_{σ_i} is true in $\mathcal{J}^{\hat{\mu}^\xi\,\hat{\nu}^\xi\,\hat{\varepsilon}^\xi}_{\hat{q}\;\hat{Q}\;\hat{E}}$

where $\hat{q}$, $\hat{Q}$ and $\hat{E}$ denote all the predicate symbols q_{F_j}, Q_{F_j} and E_{F_j} respectively.

We can now prove the following proposition.

Proposition 3. For $W_P(k,Q_{F_o}) = \{q_{F_o}(k)\} \cup \hat{W}_\sigma$, where $\hat{W}_\sigma$ satisfies I, II, and III

i) F_i is called with δ in the computation of P_ϑ for inputs ξ

$$\Leftrightarrow \vdash^{\vartheta^{\xi\,\delta}_{k\,d}}_{W_P(k,Q_{F_o})} q_{F_i}(d) \quad .$$

ii) F_i is called with δ in the computation of P_ϑ for inputs ξ, and terminates with value μ

$$\Leftrightarrow \vdash^{\vartheta^{\xi\,\delta\,\mu}_{k\,d\,e}}_{W_P(k,Q_{F_o})} Q_{F_i}(d,e) \quad .$$

Proof

i)

=>

LHS => F_i must be called at the top level of some other function variable, which in turn must have been called at the top level of another, and so on back to F_o called with ξ. Hence by repeated use of Ia),

$$\vdash^{\vartheta^{\xi\,\delta}_{k\,d}}_{W_P(k,Q_{F_o})} q_{F_i}(d) \quad .$$

<=

RHS => $\models^{\xi\,\delta}_{k\,d}$ $q_{F_i}(d)$ in $W_P(k,Q_{F_o})$

=> $\forall\hat{q}\forall\hat{Q}\forall\hat{E}[W_P(k,Q_{F_o}) \supset q_{F_i}(d)]$ true in $\mathcal{J}^{\xi\,\delta}_{k\,d}$.

By IIIa), for all $W_{\sigma_i} \epsilon \hat{W}_\sigma$, W_{σ_i} is true in $\mathcal{J}^{\hat{\mu}^\xi\,\hat{\nu}^\xi\,\hat{\varepsilon}^\xi}_{\hat{q}\;\hat{Q}\;\hat{E}}$; and $\mu^\xi_{F_o}(\xi)$.

Therefore $W_P(k,Q_{F_o})$ is true in $\mathcal{J}^{\xi\,\hat{\mu}^\xi\,\hat{\nu}^\xi\,\hat{\varepsilon}^\xi}_{k\;\hat{q}\;\hat{Q}\;\hat{E}}$

$\therefore$ $\mu^\xi_{F_i}(\delta)$

i.e., F_i is called with δ in the computation of $P_\mathcal{J}$ for inputs ξ .

ii)

=>

LHS => $\models^{\xi\,\delta}_{k\,d}$ $q_{F_i}(d)$ in $W_P(k,Q_{F_o})$ by i)

then by IIa)

$\models^{\xi\,\delta\,\mu}_{k\,d\,e}$ $Q_{F_i}(d,e)$ in $W_P(k,Q_{F_o})$

$\Leftarrow$

As for the second half of i)

$$\text{RHS} \Rightarrow \nu^{\xi}_{F_i}(\delta,\mu)$$

i.e., F_i is called with δ in computation of P_ϑ for inputs ξ, and terminates with μ.

Q.E.D.

We see that $W_P(k,Q_{F_o})$ describes the computations of P_ϑ in the desired way.

An immediate corollary is

Corollary

$$\vartheta^{\xi\,\mu}_{k\,e} \vdash_{W_P(k,Q_{F_o})} Q_{F_o}(k,e) \Leftrightarrow \text{computation of } P_\vartheta \text{ for inputs } \xi \text{ terminates with result } \mu .$$

Hence $W_P(k,Q_{F_o})$ is a logical definition of P_ϑ.

The closure of $W_P(k,Q_{F_o})$ is a universal formula and contains only extra predicate symbols. However, $\exists\hat{q}\exists\hat{Q}\exists\hat{E}W_P(k,Q_{F_o})$ is not a formalization of partial correctness of P_ϑ since it is not necessarily monotone. However, if we add an axiom, defining $W'_P(k,\phi)$ as the closure of

$$W_P(k,Q_{F_o}) \cup \{Q_{F_o}(k,x) \supset \phi(k,x)\} ,$$

then $W'_P(k,\phi)$ is still a logical definition of P_ϑ, and is clearly monotone.

By the Corollary above, for any set $\hat{\Psi}$ of relations on $|\mathcal{I}|^{n_i+1}$,

$$\forall \Psi \in \hat{\Psi} : [\exists \hat{q} \exists \hat{Q} \exists \hat{E} W'_P(k,\phi) \text{ is true in } \mathcal{I}_{k\phi}^{\xi\Psi}]$$

$$\Rightarrow \forall \xi \in |\mathcal{I}|^{n_o}, \ \forall \mu \in |\mathcal{I}| : [\nu_{F_o}^{\xi}(\xi,\mu) \Rightarrow \cap\hat{\Psi}(\xi,\mu)]$$

By condition IIIa)

$$W'_P(k,\phi) \text{ is true in } \mathcal{I}_{k\,\hat{q}\,\hat{Q}\,\hat{E}\,\phi}^{\xi\,\hat{\mu}^{\xi}\,\hat{\nu}^{\xi}\,\hat{\varepsilon}^{\xi}\,\nu_{F_o}^{\xi}}$$

$\therefore$ by monotonicity

$$\exists \hat{q} \exists \hat{Q} \exists \hat{E} W'_P(k,\phi) \text{ is true in } \mathcal{I}_{k\,\phi}^{\xi \cap \hat{\Psi}}$$

$\therefore$ $W'_P(k,\phi)$ is continuous.

Hence $\exists \hat{q} \exists \hat{Q} \exists \hat{E} W'_P(k,\phi)$ is a formalization of partial correctness of $P_{\mathcal{I}}$.

This means that $W'_P(k,\phi)$ can be used to formalize all the usual properties of $P_{\mathcal{I}}$, (including properties concerning the detection of error conditions, see Ashcroft [1]).

Manna and Pnueli [11] have formalized the partial correctness of programs very similar to the functional programs considered here. The work was done about the same time as the work presented in this section, but independently. The relationship between the two approaches is worth considering.

Relationship to the Work of Manna and Pnueli

The functional programs considered by Manna and Pnueli are slightly different from those considered here, but the principles of computation are basically the same. The difference between the two methods lies in the formulae or axioms used to describe programs. Basically, Manna and Pnueli associate a single predicate symbol, Q'_{F_i} say, with each function variable, while here we associate two symbols q_{F_i} and Q_{F_i}. If, in the axioms given in this section, all the formulae $q_{F_i}(\tau_1 \ldots \tau_n)$ were replaced by T (true) and the axioms were then simplified, the result would be the formulae of Manna and Pnueli. The reason is that $Q'_{F_i}(\tau_1 \ldots \tau_{n_i}, \tau_{n_i+1})$ has the intuitive meaning 'F_i, called with $\tau_1 \ldots \tau_{n_i}$, terminates with τ_{n_j+1}', whereas $Q_{F_i}(\tau_1 \ldots \tau_{n_i}, \tau_{n_i+1})$ has the intuitive meaning 'F_i, called with arguments $\tau_1 \ldots \tau_{n_i}$ such that $q_{F_i}(\tau_1 \ldots \tau_{n_i})$, terminates with τ_{n_i+1}'. That is $Q_{F_i}(\tau_1 \ldots \tau_{n_i+1}) \equiv q_{F_i}(\tau_1 \ldots \tau_{n_i}) \wedge Q'_{F_i}(\tau_1 \ldots \tau_{n_i+1})$. Clearly putting $q_{F_i}(\tau_1 \ldots \tau_n) \equiv T$, makes Q_{F_i} become identical to Q'_{F_i}, and it is not surprising that the simplified axioms are identical to the Manna and Pnueli formulae.

Considering the Manna-Pnueli formulae as logical definitions, (which they are, by Correspondence Theorem 2) we find that they do not describe computations in the same way. The relevant, second part of Proposition 3 holds only in the forward direction: it is possible to deduce the situations occurring in a computation, but not all the formulae deduced describe situations that occur.

According to the criteria given in the previous section, the specification of the Manna-Pnueli formulae would not constitute a logical definition of the functional language .

However, the Manna-Pnueli formula are simpler than the axioms given here, and in general they give shorter proofs of properties of programs.

Conclusion

A method of defining multi-level programs has been given, and the conditions that the axioms of such a definition must satisfy have been given in a form that enables such axioms to be checked intuitively. These conditions can be merged with the conditions for one-level language definitions in a simple way, as will be shown in the next section. Using these new conditions we will develop a logical definition of a large subset of Algol.

Section 5: ALGOL-LIKE PROGRAMS

In this section we are going to develop a logical definition of large subset of Algol 60. The features in the language will be added successively to a simple language that includes statements and specified functions (called procedures). At each stage the required modifications will be given that have to be made to the simple language definition.

The final language will have most of the features of Algol 60, including many types of statements -- assignment, conditional, jump, while, block and non-type procedures -- using expressions containing type-procedures, including boolean and array procedures.

The language will not have the 'call by name' feature, nor will it allow label and procedure parameters. However, 'side effects' of procedure calls will be possible by the use of non-local variables. There will be no input-output operations as such; programs will have certain 'input-variables' which are given values at the start of a computation, and certain of the variables in the program will be designated as output variables, whose values at the end of the computation are to be considered to be the results of the computation.

Despite these restrictions, the language will be quite a good approximation to Algol 60.

The logical definition of this language can also be used to formalize partial correctness of programs, and therefore can be used for formalizing many other properties of programs.

In the rest of this section we assume some familiarity with the constructs of Algol 60. This will allow informal descriptions of programs.

The Simple Language

We start with a simple language in which a program is a sequence of statements: assignment, condition, and jump statements. Variables are not 'typed', and there are no arrays. In these respects, the language is similar to the flowchart languages considered by Manna and Floyd. However, in the assignment statements we allow the use of procedures. A program therefore includes a set of simple procedure declarations.

Each such declaration consists of a list of formal parameters, followed by a declaration of local variables, followed by a sequence of statements (the procedure body). The name of a procedure is used like a variable in the body of the procedure on the left hand sides of assignment statements, to hold the result of the procedure call. We restrict the scope rules so that the only other variables that can occur in procedure bodies are the locally declared variables, and the formal parameters of the procedure. There can therefore be no reference to non-local variables declared outside the procedure declaration. In a similar way there can be no jump statements within a procedure body whose destinations are labelled statements outside the procedure declaration.

A program consists of a list of input variables, followed by procedure declarations and a declaration of local variables, followed by a sequence of statements (the program body), followed by an indication of the output variables. The variables that can occur in the body of the program are the input variables and the local variables.

A program P therefore looks as follows (variables are indicated by indexed letters X , Y and Z):

```
program P(X_1,...,X_{n_p});                    specification of n_p input-variables
  begin
    decl Z_1,...,Z_{m_p}                       m_p local program variables
    procedure F_1(Y_11,...,Y_1n_1);            n_1 formal parameter of F_1
      begin
        decl Z_11,...,Z_1m_1;                  m_1 local variables of F_1
        ⟨statement⟩;                           { body of F_1. Variables allowed:
          ⋮
        ⟨statement⟩                            { Y_11,...,Y_1n_0, Z_11,...,Z_1m_1 and F_1.
      end;
      ⋮
    procedure F_j(Y_j1,...,Y_jn_j);            n_j formal parameters of F_j
      begin
        decl Z_j1,...,Z_jm_j;                  m_j local variables of F_j
        ⟨statement⟩;                           { body of F_j. Variables allowed:
          ⋮
        ⟨statement⟩                            { Y_j1,...,Y_jn_j, Z_j1,...,Z_jm_j and F_j.
      end
    ⟨statement⟩;                               { body of program. Variables
    ⟨statement⟩;                               { allowed: X_1,...,X_{n_p}, Z_1,...,Z_{m_p}.
      ⋮
    ⟨statement⟩
    output(X_{α_1},...,X_{α_{n_o}})            { specification of n_o output variables,
                                                  taken from X_1,...,X_{n_p}, Z_1,...,Z_{m_p}.
  end
```

We consider $\underline{\text{output}}(X_{\alpha_1} \ldots X_{\alpha_{n_o}})$ to be a statement, but such statements can only appear at the end of program bodies.

The various other types of statements are as follows:

i) Null statement.

null

ii) Assignment statement.

$X_i := \tau$

where τ is a term as defined in the previous section (with procedure names instead of function variables).

iii) Conditional statement.

if π then ⟨statement⟩ else ⟨statement⟩

where π is a propositional term as defined in the previous section (i.e., no procedure names). The 'else ⟨statement⟩' part is optional when omitting it does not introduce ambiguity.

iv) Jump statement.

goto L_i

where L_i is a label. Any statement can be preceded by one or more labels, each followed by a colon. In any procedure or program body, any label occurring in a jump statement occurs exactly once labelling a statement.

To simplify the later logical definition we stipulate that the last statement in the sequence forming a procedure body is null.

We can now give an example of a program. The statements have been numbered for later reference. The numbers are not part of the program.

```
program sort(list);
   begin
      decl result;
      procedure merge(sortlist,atom);
         begin
        (8) if ¬ null(sortlist) ∧ lessp(atom,car(sortlist))
               then (9) merge := cons(car(sortlist),merge(cdr(shortlist),atom))
               else (10) merge := cons(atom,sortlist);
        (11) null
         end
   (1) result := NIL;
 (2) L: if null(list) then (3) go to out;
   (4) result := merge(result,car(list));
   (5) list := cdr(list);
   (6) go to L;
(7) out: output(result)
   end
```

$|\mathcal{I}|$ consists of atoms and lists of atoms. The lisp functions and predicates have their usual meanings, and 'lessp' is some relation that totally orders the atoms.

Execution of programs

The execution of these simple programs conforms with the usual Algol meaning, with the following restrictions:

i) On calling a procedure, the parameters are passed by value only. Together with the scope restrictions given previously, this means that calling a procedure produces no side effects. Hence in the evaluation of terms, e.g. $f_1(\tau_1,\tau_2)$, it does not matter which subterm

τ_1 or τ_2 is evaluated first. However, we will allow side effects later, so we will stipulate that

ii) Terms are evaluated left to right, i.e., τ_1 before τ_2.

In most sensible programs, the value of a variable is not used before the variable has been assigned a value by an assignment statement. However, when a variable is declared it must have some value, and we stipulate that

iii) the initial value must be the same for all declared variables. This holds for the name of a procedure used to return the value of the procedure (i.e., this value will be returned by any call of a procedure in which no statement is executed that assigns a value to the name of the procedure). We assume this special value corresponds to a special constant β in the basis symbols of $\mathcal{S}$.

We do not intend to give a description of the execution of programs, but to simplify the later logical definition we assume

a) There is a known correspondence between variables occurring in statements and their first occurrences in parameter lists or variable declarations. This means that there is no difficulty in renaming all variables in the program without changing its computations.

b) There is a known correspondence between labels occurring in jump statements and the statements that are the destinations of the jumps.

c) Every statement has a 'successor' statement, defined as follows:

i) For statements comprising the sequence of statements that is a procedure or program body, the successor statement is simply the next statement in the sequence. For the last statement in the sequence, i.e., null or output, the successor will be denoted by ε .

ii) For statements contained in other statements (in this case, in conditional statements) the successor statement is the successor of the smallest containing statement.

We will not give examples of computations, but merely remark that the above program 'sort' sorts lists according to the relation 'lessp' on the elements of the lists.

Logical definition of the simple language

We intend to amalgamate the two previous definition techniques, for the one-level language and for functional programs, into a definition of the simple language. This means introducing three types of predicate symbols: ϕ_σ , indicating that computation has reached statement σ , q_{F_i} , indicating the call of procedure F_i , and Q_{F_i} , indicating that F_i is called and returns a value. The symbols q_{F_i} and Q_{F_i} are n_i-ary and n_i+1-ary as in the previous section. For ϕ_σ , where

statement σ is in the body of procedure F_i {the program P}, we define the numbers $\sigma' = n_i\{n_p\}$ and $\sigma'' = (n_i + m_i + 1)\{(n_p + m_p)\}$ */. Then ϕ_σ is $\sigma' + \sigma''$ -ary. σ'' is the number of variables in scope for σ, and in fact we can map these variables into the integers 1 to σ'' according to the order of their first occurrences in parameter lists or declarations, e.g. for

```
program Prg(foo,baz);
   begin
      decl A,B,C;
      procedure F(D,E);
         begin
            decl G;
            σ1; σ2; σ3; null
         end
      σ4;
      σ5;
      output(B,C)
   end
```

in the program body,

$\sigma_4' = 2$, $\sigma_4'' = 5$, and the variables are ordered

foo, baz, A, B, C;

in the body of F,

$\sigma_2' = 2$, $\sigma_2'' = 4$, and the variables are ordered

F, D, E, G

*/ As in the previous outline of a program, n_i is the number of formal parameters of F_i, and m_i is the number of locally declared variables.

With the program P itself we associate predicate symbols

$$q_P \ (n_p\text{-ary}) \quad \text{and} \quad Q_P \ (n_p + n_o\text{-ary}) \ .$$

We are going to construct formulas using these predicate symbols, and the construction of these formulas will be more obvious if the symbols are considered to have the following meanings:

$q_{F_i}(\tau_1,\ldots,\tau_{n_i})$ means 'procedure F_i is called with arguments $\tau_1,\ldots,\tau_{n_i}$'.

$Q_{F_i}(\tau_1,\ldots,\tau_{n_i},\tau_{n_i+1})$ means 'procedure F_i is called with arguments $\tau_1\ldots\tau_{n_i}$ and terminates with value τ_{n_i+1}'.

$\phi_\sigma(t_1,\ldots,t_{\sigma'},\tau_1,\ldots,\tau_{\sigma''})$ means 'computation reaches statement σ (in some procedure {program} body) with variable values $\tau_1\ldots\tau_{\sigma''}$ when the procedure {program} was called with arguments {inputs} $t_1\ldots t_{\sigma'}$'.

The formulas are constructed as follows.

We assume that X_i represents the i-th variable in the ordering for a particular procedure declaration or program.

I) With each statement σ whose successor is σ_2, and $\sigma_2 \neq \varepsilon$, we associate a formula W_σ:

i) Null statement:

σ:- null

W_σ:- $\phi_\sigma(y_1,\ldots,y_{\sigma'},x_1,\ldots,x_{\sigma''}) \supset \phi_{\sigma_2}(y_1,\ldots,y_{\sigma_2'},x_1,\ldots,x_{\sigma_2''})$

ii) Assignment statement: W_σ is best illustrated by examples

a) σ :- $X_j := f_1(X_i, f_1(b_1, X_k))$

W_σ :- $\phi_\sigma(y_1 \cdots y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset$

$\phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, x_1, \ldots, x_{j-1}, f_1(x_i, f_1(b_1, x_k)), \ldots, x_{\sigma_2''})$

b) σ:- $X_j := F_1(X_i, f_2(X_j))$

W_σ:- $\phi_\sigma(y_1, \ldots, y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset$

$\{q_{F_1}(x_i, f_2(x_j)) \wedge$

$[Q_{F_1}(x_i, f_2(x_j), z_1) \supset$

$\phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_1, \ldots, x_{\sigma_2''})]\}$

c) σ:- $X_j := f_1(F_1(F_2(X_i), F_2(X_j)), F_2(X_k))$

W_σ:- $\phi_\sigma(y_1, \ldots, y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset$

$\{q_{F_2}(x_i) \wedge$

$[Q_{F_2}(x_i, z_1) \supset$

$\{q_{F_2}(x_j) \wedge$

$[Q_{F_2}(x_j, z_2) \supset$

$\{q_{F_1}(z_1, z_2) \wedge$

$[Q_{F_1}(z_1, z_2, z_3) \supset$

$\{q_{F_2}(x_k) \wedge$

$[Q_{F_2}(x_k, z_4) \supset$

$\phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1, \ldots, x_{j-1}, f_1(z_3, z_4), \ldots, x_{\sigma_2''})]\}]\}]\}]\}$.

This example illustrates the left-to-right rule for evaluating terms.

The construction of W_σ for all assignment statements should be obvious from these examples.

iii) Conditional statements

a) σ:- $\underline{\text{if}}\ \pi(X_{\beta_1},\ldots,X_{\beta_j})\ \underline{\text{then}}\ \sigma_0\ \underline{\text{else}}\ \sigma_1$

where σ_0,σ_1 are statements, and $\pi(X_{\beta_1},\ldots,X_{\beta_j})$ is a propositional term $(X_{\beta_i} \in \{X_1,\ldots,X_{\sigma''}\},\ i = 1\ldots j)$

W_σ:- $\phi_\sigma(y_1,\ldots,y_{\sigma'},x_1\ldots x_{\sigma''}) \supset$

$\underline{\text{if}}\ \pi(x_{\beta_1},\ldots,x_{\beta_j})\ \underline{\text{then}}\ \phi_{\sigma_0}(y_1\ldots y_{\sigma_0'},x_1\ldots x_{\sigma_0''})$

$\underline{\text{else}}\ \phi_{\sigma_1}(y_1\ldots y_{\sigma_1'},x_1\ldots x_{\sigma_1''})$ */

b) σ:- $\underline{\text{if}}\ \pi(X_{\beta_1},\ldots,X_{\beta_j})\ \underline{\text{then}}\ \sigma_0$

W_σ:- $\phi_\sigma(y_1\ldots y_{\sigma'},x_1\ldots x_{\sigma''}) \supset$

$\underline{\text{if}}\ \pi(x_{\beta_1},\ldots,x_{\beta_j})\ \underline{\text{then}}\ \phi_0(y_1,\ldots,y_{\sigma_0'},x_1\ldots x_{\sigma_0''})$

$\underline{\text{else}}\ \phi_{\sigma_2}(y_1,\ldots,y_{\sigma_2'},x_1,\ldots,x_{\sigma_2''})$

---

*/ For formulas, $\underline{\text{if}}$ P $\underline{\text{then}}$ A $\underline{\text{else}}$ B means $[[P \supset A] \wedge [\neg P \supset B]]$.

iv) Jump statements

σ:- <u>goto</u> L_i

where label L_i corresponds to statement σ_1

W_σ:- $\phi_\sigma(y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''}) \supset$

$\phi_{\sigma_1}(y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''})$

II) For statement σ whose successor is ε, there are two cases to consider:

i) if σ is the last statement in the body of procedure F_i, i.e., <u>null</u>, then with this procedure declaration we associate the axioms

W_{F_i}:- $\{q_{F_i}(y_1, \ldots, y_{n_i}) \supset \phi_{\sigma_0}(y_1, \ldots, y_{n_i}, \beta, y_1, \ldots, y_{n_i}, \beta \ldots \beta),$

$\phi_\sigma(y_1, \ldots, y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset Q_{F_i}(y_1 \cdots y_{n_i}, x_1)\}$

where σ_0 is the first statement in the body of F_i. (Note that $\sigma_0' = \sigma' = n_i$.)

ii) if σ is the last statement in the program P, i.e.,

<u>output</u>$(X_{\alpha_1} \ldots X_{\alpha_{n_o}})$, where $X_{\alpha_i} \epsilon \{X_1 \ldots X_{\sigma''}\}$, $i = 1 \ldots n_o$,

then with the program we associate the axioms

W_P:- $\{q_P(y_1 \cdots y_{n_p}) \supset \phi_{\sigma_0}(y_1, \ldots, y_{n_p}, y_1, \ldots, y_{n_p}, \beta \ldots \beta),$

$\phi_\sigma(y_1, \ldots, y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset Q_P(y_1, \ldots, y_{n_p}, x_{\alpha_1} \ldots x_{\alpha_{n_o}})\}$

where σ_0 is the first statement in the body of P.

(Note that $\sigma_0' = \sigma' = n_p$.)

III) We then define the set of axioms $W_P(k,Q_P)$ */ as

$$\{q_P(k), W_P\} \cup \{W_\sigma \mid \sigma \text{ in } P\} \cup \{W_{F_i} \mid F_i \text{ in } P\} .$$

For example, $W_P(k,Q_P)$ for the program 'sort' is given below. Statements are referred to by the numbers in the example, and procedure 'merge' is referred to as F . Axioms are separated by semi-colons.

$W_P(k,Q_P)$:- $q_P(k_1)$;

$q_P(y_1) \supset \phi_1(y_1,y_1,\beta)$;

$\phi_7(y_1,x_1,x_2) \supset Q_P(y_1,x_2)$;

$\phi_1(y_1,x_1,x_2) \supset \phi_2(y_1,x_1,\text{NIL})$;

$\phi_2(y_1,x_1,x_2) \supset \underline{\text{if}}\ \text{null}(x_1)\ \underline{\text{then}}\ \phi_3(y_1,x_1,x_2)$

$\underline{\text{else}}\ \phi_4(y_1,x_1,x_2)$

$\phi_3(y_1,x_1,x_2) \supset \phi_7(y_1,x_1,x_2)$;

$\phi_4(y_1,x_1,x_2) \supset \{q_F(x_2,\text{car}(x_1)) \wedge$

$[Q_F(x_2,\text{car}(x_1),z_1) \supset \phi_5(y_1,x_1,z_1)]\}$;

$\phi_5(y_1,x_1,x_2) \supset \phi_6(y_1,\text{cdr}(x_1),x_2)$;

$\phi_6(y_1,x_1,x_2) \supset \phi_2(y_1,x_1,x_2)$;

---

*/ $k = (k_1 \ldots k_{n_p})$ are constant symbols.

$$q_F(y_1,y_2) \supset \phi_8(y_1,y_2,\beta,y_1,y_2);$$

$$Q_{11}(y_1,y_2,x_1,x_2,x_3) \supset Q_F(y_1,y_2,x_1);$$

$$\phi_8(y_1,y_2,x_1,x_2,x_3) \supset \underline{\text{if}}\ \neg\, \text{null}(x_2) \wedge \text{lessp}(x_3,\text{car}(x_2))$$
$$\underline{\text{then}}\ \phi_9(y_1,y_2,x_1,x_2,x_3)$$
$$\underline{\text{else}}\ \phi_{10}(y_1,y_2,x_1,x_2,x_3);$$

$$\phi_9(y_1,y_2,x_1,x_2,x_3) \supset \{q_F(\text{cdr}(x_2),x_3) \wedge$$
$$[Q_F(\text{cdr}(x_2),x_3,z_2) \supset$$
$$\phi_{11}(y_1,y_2,\text{cons}(\text{car}(x_1),z_2),x_2,x_3)]\}$$

$$\phi_{10}(y_1,y_2,x_1,x_2,x_3) \supset \phi_{11}(y_1,y_2,\text{cons}(x_2,x_1),x_2,x_3);$$

These axioms can clearly be simplified so that $W_P(k,Q_P)$ becomes

$$q_P(k_1);$$

$$q_P(y_1) \supset \phi_2(y_1,y_1,\text{NIL});$$

$$\phi_2(y_1,x_1,x_2) \supset \underline{\text{if}}\ \text{null}(x_1)$$
$$\underline{\text{then}}\ Q_P(y_1,x_2)$$
$$\underline{\text{else}}\ \{q_F(x_2,\text{car}(x_1)) \wedge$$
$$[Q_F(x_2,\text{car}(x_1),z_1) \supset$$
$$\phi_2(y_1,\text{cdr}(x_1),z_1)]\};$$

$$q_F(x_1,x_2) \supset \underline{\text{if}} \neg \text{null}(x_1) \wedge \text{lessp}(x_2,\text{car}(x_1))$$

$$\underline{\text{then}} \{q_F(\text{cdr}(x_1),x_2) \wedge$$

$$[Q_F(\text{cdr}(x_1),x_2,z_2) \supset$$

$$Q_F(x_1,x_2,\text{cons}(\text{car}(x_1),z_2))]\}$$

$$\underline{\text{else}}\ Q_F(x_1,x_2,\text{cons}(x_2,x_1));$$

Clearly the construction of $W_P(k,Q_P)$ uses the techniques developed for both the one-level language and for functional programs. We will not go through the proof that $W_P(k,Q_P)$ is a logical definition of $P_\mathcal{J}$ that describes computations of $P_\mathcal{J}$, since it is similar to previous proofs, only longer. We will simply give the conditions that $W_P(k,Q_P)$ satisfies and state the relevant proposition.

Any computation of a procedure or program body consists of a sequence of computations of statements from the procedure or program body. When we talk of computations of such statements, the 'next' statement is the statement (in the same procedure or program body) whose computation follows the computation of the statement considered. This is not always the same as the successor statement, for example for jump statements or conditional statements.

The conditions on

$$\hat{W}_P = \{W_\sigma | \sigma \text{ in } P\} \cup \{W_{F_i} | F_i \epsilon P\}$$

are as follows.

I) If for computation of statement σ, for variable values $(\eta_1 \cdots \eta_{\sigma''}) = \eta$ the next statement σ_1 is reached with variable values $(\mu_1 \cdots \mu_{\sigma_1''}) = \mu$
then for all $\delta = (d_1 \cdots d_{\sigma'}) \in |\mathfrak{d}|^{\sigma'}$

$$\mathfrak{d}^{\delta\,\eta\,\mu}_{d\,g\,h} \vdash_{\phi_\sigma(d,g),\hat{W}_P} \phi_{\sigma_1}(d,h) \quad .$$

II) If for computation of statement σ, for variable values $(\eta_1 \cdots \eta_{\sigma''}) = \eta$, procedure F_i is called (at the top level, i.e., σ contains F_i) with arguments $(\zeta_1, \ldots, \zeta_{n_i}) = \zeta$
then for all $\delta = (\delta_1, \ldots, \delta_{\sigma'}) \in |\mathfrak{d}|^{\sigma'}$

$$\mathfrak{d}^{\delta\,\eta\,\zeta}_{d\,g\,h} \vdash_{\phi_\sigma(d,g),\hat{W}_P} q_{F_i}(h) \quad .$$

III) If computation of F_i called with $(\zeta_1 \cdots \zeta_{n_i}) = \zeta$, reaches statement σ in the body of F_i with variable values $(\eta_1 \cdots \eta_{\sigma''}) = \eta$
then

$$\mathfrak{d}^{\zeta\,\eta}_{h\,g} \vdash_{q_{F_i}(h),\hat{W}_P} \phi_\sigma(h,g) \quad .$$

IV) If computation of F_i called with $(\zeta_1 \cdots \zeta_{n_i}) = \zeta$ terminates with γ
then

$$\mathfrak{d}^{\zeta\,\gamma}_{h\,e} \vdash_{q_{F_i}(h),\hat{W}_P} Q_{F_i}(h,e) \quad .$$

V) For all statements σ in procedure F_j {program P}, W_σ makes sense if we precede it by 'for all sets of computations of σ' and we read $q_{F_i}(\tau_1 \ldots \tau_{n_i})$ as
'in one computation, F_i is called with $\tau_1 \ldots \tau_{n_i}$',
$Q_{F_i}(\tau_1 \ldots \tau_{n_i}, \tau_{n_i+1})$ is read as
'in one computation, F_i is called with $\tau_1 \ldots \tau_{n_i}$ and terminates with τ_{n_i+1}',
$\phi_{\sigma_i}(t_1, \ldots, t_{\sigma'}, \tau_1, \ldots, \tau_{\sigma''})$ is read as
'in one computation, which is in the computation of F_j {P} for arguments {inputs} $t_1, \ldots, t_{\sigma'}$, σ_i is computed for variable values $\tau_1, \ldots, \tau_{\sigma''}$'.

VI) For all procedures F_i in P, W_{F_i} makes sense if we precede it by 'for all sets of computations of F_i' and the predicate symbols are read as in V) above.

It is clear that $\hat{W}_P$ satisfies the above conditions (conditions I-IV require induction on lengths of computations as in the previous section, but are quite straightforward). From these conditions we can prove the following proposition.

Proposition 4. If $W_P(k, Q_P)$ satisfies I - VI: in computation of P for inputs $(\xi_1 \ldots \xi_{n_p}) = \xi$,

a) statement σ in the program body is executed for variable values $(\eta_1 \cdots \eta_{\sigma''}) = \eta$

b) F_i is called with $(\delta_1 \cdots \delta_{n_i}) = \delta$

c) F_i is called with $(\delta_1 \cdots \delta_{n_i}) = \delta$ and
   i) statement σ in the body of F_i is executed for variable values $(\eta_1 \cdots \eta_{\sigma''}) = \eta$
   ii) computation of F_i terminates with γ

d) computation of P terminates with output values $(\mu_1 \cdots \mu_{n_o}) = \mu$

<u>if and only if</u>

a) $\mathfrak{I}^{\xi\,\eta}_{k\,g} \vdash_{W_P(k,Q_P)} \phi_\sigma(k,g)$

b) $\mathfrak{I}^{\xi\,\delta}_{k\,d} \vdash_{W_P(k,Q_P)} q_{F_i}(d)$

c)
   i) $\mathfrak{I}^{\xi\,\delta\,\eta}_{k\,d\,g} \vdash_{W_P(k,Q_P)} \phi_\sigma(d,g)$
   ii) $\mathfrak{I}^{\xi\,\delta\,\gamma}_{k\,d\,e} \vdash_{W_P(k,Q_P)} Q_{F_i}(d,e)$

d) $\mathfrak{I}^{\xi\,\mu}_{k\,h} \vdash_{W_P(k,Q_P)} Q_P(k,h)$

Proof is similar to the proofs of the previous three propositions.

We see that $W_P(k,Q_P)$ is a logical definition of P_ϑ and it describes the computations of P_ϑ in the desired way.

It can also be easily shown, as in the previous section, that $W_P(k,Q_P)$ is both monotone and continuous, and therefore $\exists Q\tilde{W}_P(k,Q_P)$ is a formalization of partial correctness (where Q denotes all the extra predicate symbols except Q_P, and $\tilde{W}_P(k,Q_P)$ is the closure of $W_P(k,Q_P)$). We can therefore use $W_P(k,Q_P)$ to formalize properties of P such as termination, correctness, etc.

We can now add extra features to the language and modify $W_P(k,Q_P)$ accordingly. It is clear that after each modification, $\hat{W}_P$ still satisfies I - VI (possibly modified slightly) and so $W_P(k,Q_P)$ still is a logical definition.

Extensions of the Simple Language

1. While statements

We add statements of the form

σ:- while π do σ_1

where π is a propositional term, and σ_1 is a statement.

Let the successor of σ be σ_2 ;

$$W_\sigma :- \ \phi_\sigma(y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''}) \supset$$
$$\underline{\text{if}}\ \pi\ \underline{\text{then}}\ \phi_{\sigma_1}(y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''})$$
$$\underline{\text{else}}\ \phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1 \cdots x_{\sigma_2''})$$

and we stipulate that the successor of σ_1 is σ.

2. Conditional expressions

We modify the definition of assignment statements by allowing expressions on the right hand sides.

An expression is a term or is of the form

$\underline{\text{if}}\ \pi\ \underline{\text{then}}\ \tau_1\ \underline{\text{else}}\ \tau_2$

where τ_1, τ_2 are expressions, and π is a propositional term.

We will illustrate the construction of W_σ by examples as before -- the successor of σ is σ_2.

i) $\sigma :- \ X_j := \underline{\text{if}}\ \pi(X_{\beta_1} \cdots X_{\beta_k})\ \underline{\text{then}}\ f_2(F_1(f_1(X_i)), X_k)$
$\underline{\text{else}}\ F_2(X_i, X_j)$

$$W_\sigma :- \ \phi_\sigma(y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''}) \supset$$
$$\underline{\text{if}}\ \pi(x_{\beta_1} \cdots x_{\beta_k})$$
$$\underline{\text{then}}\ \{q_{F_1}(f_1(x_i)) \wedge$$
$$[Q_{F_1}(f_1(x_i), z_1) \supset \phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1 \cdots x_{j-1}, f_2(z_1, x_k), \ldots, x_{\sigma_2''})]\}$$
$$\underline{\text{else}}\ \{q_{F_2}(x_i, x_j) \wedge$$
$$[Q_{F_2}(x_i, x_j, z_2) \supset \phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_2 \cdots x_{\sigma_2''})]\}.$$

ii) σ:- $X_j := F_2(\underline{\text{if}}\ \pi_1\ \underline{\text{then}}\ b_1\ \underline{\text{else}}\ f_1(F_1(X_j)),$

$\underline{\text{if}}\ \pi_2\ \underline{\text{then}}\ X_j\ \underline{\text{else if}}\ \pi_3\ \underline{\text{then}}\ F_1(X_k)$

$\underline{\text{else}}\ b_2)$

W_σ:- $\phi_\sigma(y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''}) \supset$

$\underline{\text{if}}\ \pi_1\ \underline{\text{then if}}\ \pi_2\ \underline{\text{then}}$

$\{q_{F_2}(b_1, x_j) \wedge$

$[Q_{F_2}(b_1, x_j, z_1) \supset \phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_1, \ldots, x_{\sigma_2''})]\}$

$\underline{\text{else if}}\ \pi_3\ \underline{\text{then}}$

$\{q_{F_1}(x_k) \wedge$

$[Q_{F_1}(x_k, z_2) \supset$

$\{q_{F_2}(b_1, z_2) \wedge$

$[Q_{F_2}(b_1, z_2, z_3) \supset \phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, x_1 \cdots x_{j-1}, z_3 \cdots x_{\sigma_2''})]\}]\}$

$\underline{\text{else}}\ \{q_{F_1}(b_1, b_2) \wedge$

$[Q_{F_1}(b_1, b_2, z_4) \supset$

$\phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_4 \cdots \sigma_2'')]\}$

$\underline{\text{else}}\ \{q_{F_1}(x_j) \wedge$

$[Q_{F_1}(x_j, z_5) \supset$

$\underline{\text{if}}\ \pi_2\ \underline{\text{then}}$

$\{q_{F_1}(f_1(z_5), x_j) \wedge$

$[Q_{F_1}(f_1(z_5), x_j, z_6) \supset$

$\phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_6, \ldots, x_{\sigma_2''})]\}$

$$\underline{\text{else if}}\ \pi_3\ \underline{\text{then}}$$

$$\{q_{F_1}(x_k) \wedge$$

$$[Q_{F_1}(x_k, z_7) \supset$$

$$\{q_{F_1}(f_1(z_5), z_7) \wedge$$

$$[Q_{F_1}(f_1(z_5), z_7, z_8) \supset$$

$$\phi_{\sigma_2}(y_1 \cdots y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_8, \ldots, x_{\sigma_2''})]\}]\}$$

$$\underline{\text{else}}\ \{q_{F_1}(f_1(z_5), b_2) \wedge$$

$$[Q_{F_1}(f_1(z_5), b_2, z_9) \supset$$

$$\phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, x_1, \ldots, x_{j-1}, z_9, \ldots x_{\sigma_2''})]\}]\}\ .$$

The construction of W_σ is clearly lengthy but straightforward.

3. Modification of propositional terms

We are going to allow general expressions in propositional terms instead of simple terms and allow *if then else* as a logical connective. This means introducing procedure calls, the *if then else* construction and a left to right evaluation rule for the generalized propositional terms (known as propositional expressions). We shall give examples of W_σ for conditional statements formed with such propositional expressions. The formulae for while statements are similar.

i) σ :- *if* $[P_j(F_1(X_j)) \wedge \neg P_j(F_1(X_k))]$ *then* σ_1 *else* σ_2

W_σ:- $\phi_\sigma(y_1,\ldots,y_{\sigma'},x_1,\ldots,x_{\sigma''}) \supset$

$\{q_{F_1}(x_j) \wedge$

$[Q_{F_1}(x_j,z_1) \supset$

$\underline{\text{if}}\ P_j(z_1)\ \underline{\text{then}}$

$\{q_{F_1}(x_k) \wedge$

$[Q_{F_1}(x_k,z_2) \supset$

$\underline{\text{if}} \neg P_j(z_2)\ \underline{\text{then}}$

$\phi_{\sigma_1}(y_1\cdots y_{\sigma_1'},x_1,\ldots,x_{\sigma_1''})$

$\underline{\text{else}}\ \phi_{\sigma_2}(y_1\cdots y_{\sigma_2'},x_1,\ldots,x_{\sigma_2''})]\}$

$\underline{\text{else}}\ \phi_{\sigma_2}(y_1\cdots y_{\sigma_2'},x_1\cdots x_{\sigma_2''})]\}$

ii) σ:- $\underline{\text{if}}\ \underline{\text{if}}\ P_j(F_1(X_i))\ \underline{\text{then}}\ [P_k(F_1(X_j)) \vee P_k(f_1(F_1(X_k)))]$

$\underline{\text{else}}\ P_j(\underline{\text{if}}\ P_k(X_k)\ \underline{\text{then}}\ b_1\ \underline{\text{else}}\ X_j)$

$\underline{\text{then}}\ \sigma_1\ \underline{\text{else}}\ \sigma_2$

W_σ:- $\phi_\sigma(y_1\cdots y_{\sigma'},x_1\cdots x_{\sigma''}) \supset$

$\{q_{F_1}(x_i) \wedge$

$[Q_{F_1}(x_i,z_1) \supset$

$\underline{\text{if}}\ P_j(z_1)\ \underline{\text{then}}$

$\{q_{F_1}(x_j) \wedge$

$[Q_{F_1}(x_j,z_2) \supset$

$$\underline{\text{if}}\ P_k(z_2)\ \underline{\text{then}}\ \phi_{\sigma_1}(y_1\cdots y_{\sigma_1'},x_1\cdots x_{\sigma_1''})$$

$$\underline{\text{else}}\ \{q_{F_1}(x_k)\ \wedge$$

$$[Q_{F_1}(x_k,z_3)\ \supset$$

$$\underline{\text{if}}\ P_k(f_1(z_3))\ \underline{\text{then}}\ \phi_{\sigma_1}(y_1\cdots y_{\sigma_1'},x_1\cdots x_{\sigma_1''})$$

$$\underline{\text{else}}\ \phi_{\sigma_2}(y_1\cdots y_{\sigma_2'},x_1\cdots x_{\sigma_2''})]]]\}$$

$$\underline{\text{else if}}\ P_k(x_k)\ \underline{\text{then}}$$

$$\underline{\text{if}}\ P_j(b_1)\ \underline{\text{then}}\ \phi_{\sigma_1}(y_1\cdots y_{\sigma_1'},x_1\cdots x_{\sigma_1''})$$

$$\underline{\text{else}}\ \phi_{\sigma_2}(y_1\cdots y_{\sigma_2'},x_1\cdots x_{\sigma_2''})$$

$$\underline{\text{else if}}\ P_j(x_j)\ \underline{\text{then}}\ \phi_{\sigma_1}(y_1\cdots y_{\sigma_1'},x_1\cdots x_{\sigma_1''})$$

$$\underline{\text{else}}\ \phi_{\sigma_2}(y_1\cdots y_{\sigma_2'},x_1\cdots x_{\sigma_2''})]\}\ .$$

4. Blocks

We allow blocks as a new type of statement.

A block is a sequence of statements, preceded (optionally) by local variable and procedure declarations. The last statement in the sequence is null.

e.g.

```
begin
   decl X1...Xi;
   procedure F1(Y1...Yj);
      begin
         :
      end;
   ⟨statement⟩;
      :
   ⟨statement⟩;
   null
end
```

We generalize procedure declarations so that they each consist of a formal parameter list followed by a block.

Clearly blocks and procedure declarations can now be nested to any depth, and the scope rules get more complicated. We still do not allow reference to non-local variables in procedure bodies, so that the variables in scope for any particular statement are simply all the variables declared in enclosing blocks out to the smallest enclosing procedure declaration {or the program itself} plus the formal parameters and name {input variables} of this procedure {program}.

e.g. for
```
program P(X1);
   begin
      decl X2;
      procedure F1(X3);
         begin
            decl X4;
            procedure F2(X5);
               begin
                  decl X6;
                  σ1;
                  σ2;
                  begin
                     decl X7;
                     σ3;
                     begin
                        decl X8;
                        σ4;
                        null
                     end;
                     null
                  end;
                  σ5;
                  null
               end;
            σ6;
```

```
            begin
               decl X9;
               σ7;
               null
            end;
            null
         end;
      σ8;
      output(X2)
   end
```

we give the variables in scope (in correct order) for statements of P as follows:

σ_1, σ_2 : F_2, X_5, X_6;

σ_3 : F_2, X_5, X_6, X_7;

σ_4 : F_2, X_5, X_6, X_7, X_8;

σ_5 : F_2, X_5, X_6;

σ_6 : F_1, X_3, X_4;

σ_7 : F_1, X_3, X_4, X_9;

σ_8 : X_1, X_2.

The formulae $\hat{W}_P$ are based on the fact that σ'' is the number of variables in scope for σ. Therefore we have to redefine σ' and σ'' :

σ' = (number of formal parameters of the smallest procedure declaration enclosing σ),

and

σ'' = $1+\sigma'$ + (number of local variables declared in blocks, within this procedure declaration, enclosing σ).

If σ is not enclosed by a procedure declaration, σ' is the number of input variables and $\sigma'' = \sigma' +$ (number of local variables declared in blocks enclosing σ).

The modifications of the definitional axioms are then quite simple. Let σ be the block

```
begin
   decl X_β1,...,X_βj;
   procedure  :
              :
   σ0
   :
   σ1
end
```

i.e., σ_0 is the first statement and σ_1 (i.e., null) is the last statement, and let the successor of σ be σ_2 . We simply define the successor of σ_1 to be σ_2 , and associate with σ the axiom

$$W_\sigma :- \ \phi_\sigma(y_1,\ldots,y_{\sigma'},x_1\ldots x_{\sigma''}) \supset \phi_{\sigma_0}(y_1\ldots y_{\sigma'},x_1,\ldots,x_{\sigma''},\underbrace{\beta,\ldots,\beta}_{j \text{ times}})$$

(Note that $\sigma''_0 = \sigma'' + j$, by definition)

For any statement σ_i in σ which causes execution to leave σ (i.e., σ_i is σ_1 or is a jump statement), the axiom W_{σ_i} will show that the values of the j local variables are lost on leaving σ . This is because for the statement σ_j that is reached, σ''_j is at least j less than σ''_i , and it is the rightmost argument values (corresponding to the local variables) that are dropped.

e.g. for the following fragment of a program

```
        procedure F1(X2);
           begin
              decl X3;
              :
σ1 ----- L:
              begin
                 decl X4;
                 :
σ2 ---------- begin
                    decl X5;
σ3 ---------- N:
                    :
σ4 ---------- goto M;
                    :
σ5 ---------- goto L;
                    :
σ6 ---------- null
                 end
σ7 --------- K:
                 :
σ8 --------- M:
                 :
                 null
              end
              :
              null
           end
```

some of the axioms are as follows:

$$W_{\sigma_2} :- \phi_{\sigma_2}(y_1, x_1, x_2, x_3, x_4) \supset \phi_{\sigma_3}(y_1, x_1, x_2, x_3, x_4, \beta)$$

$$W_{\sigma_4} :- \phi_{\sigma_4}(y_1, x_1, x_2, x_3, x_4, x_5) \supset \phi_{\sigma_8}(y_1, x_1, x_2, x_3, x_4)$$

$$W_{\sigma_5} :- \phi_{\sigma_5}(y_1, x_1, x_2, x_3, x_4, x_5) \supset \phi_{\sigma_1}(y_1, x_1, x_2, x_3)$$

$$W_{\sigma_6} :- \phi_{\sigma_6}(y_1, x_1, x_2, x_3, x_4, x_5) \supset \phi_{\sigma_7}(y_1, x_1, x_2, x_3, x_4)$$

(The procedure name F_1 corresponds to x_1 .)

5. 'Side-effects'

We allow statements in procedure bodies to use non-local variables, i.e., the variables declared in blocks enclosing the procedure declaration, and the formal parameter and procedure names of other procedure declarations that enclose the one in question.

This means that a procedure F_i is a function not just of n_i arguments, but of $n_i + g_i$ arguments, where g_i is the number of non-local, or 'global', variables to which F_i has access. For any F_i, g_i is a fixed number, namely the number of input variables plus the number of variables and formal parameters (and procedure names) in blocks and procedure declarations enclosing the declaration of F_i. In fact there is a mapping of these g_i global variables into the integers 1 to g_i according to the order of their first occurrences in parameter lists and variable declarations. A given variable is mapped into the SAME number by the appropriate orderings for all the procedures that have this variable as a global. That is, we can assign each variable and procedure name a number, and whenever this variable or procedure name is a global of a procedure, its position in the ordering of globals is just this number.

The effect of F_i is no longer limited to returning a single value, but it may now change any of the g_i global variables.

We must therefore adjust the number of argument places of the predicate symbols q_{F_i} and Q_{F_i} which have to describe the effect of F_i; q_{F_i} becomes g_i+n_i-ary and Q_{F_i} becomes $g_i+n_i+g_i+1$-ary.

For ϕ_σ we redefine σ' and σ''. If the smallest procedure declaration enclosing σ is that of F_i, then we put $\sigma' = g_i + n_i$. Then, as before, $\sigma'' = 1 + \sigma' +$ (number of variables declared in blocks containing σ, contained in the declaration of F_i). If there is no such procedure declaration enclosing σ, then σ' and σ'' are not altered. In both cases ϕ_σ is $\sigma' + \sigma''$-ary.

This definition of σ'' ensures that the number of variables that can appear in σ is σ'', and the order of appearance of these variables in parameter lists and declarations maps them into the integers 1 to σ''.

The modifications to $W_P(k,Q_P)$ to deal with side effects are as follows:

a) The axioms W_σ are defined exactly as before (using the new definitions of σ' and σ'') except for statements that include procedure symbols (procedure calls). The method of construction of W_σ for these latter statements can easily be inferred from the following example, which was used previously in the original specification of W_σ:

$$\sigma{:-}\ X_j := f_1(F_1(F_2(X_i),F_2(X_j)),F_2(X_k)) .$$

Note that both g_1 and g_2 (for F_1 and F_2) must be less than or equal to σ'' because of the scope rules. We assume $j > g_1 > g_2$.

$$W_\sigma :- \ \phi_\sigma(y_1 \cdots y_{\sigma'}, x_1, \ldots, x_{\sigma''}) \supset$$
$$\{q_{F_2}(x_1, \ldots, x_{g_2}, x_i) \wedge$$
$$[Q_{F_2}(x_1 \cdots x_{g_2}, x_i, v_{11}, \ldots, v_{1g_2}, z_1) \supset$$
$$\{q_{F_2}(v_{11}, \ldots, v_{1g_2}, x_j) \wedge$$
$$[Q_{F_2}(v_{11}, \ldots, v_{1g_2}, x_j, v_{21}, \ldots, v_{2g_2}, z_2) \supset$$
$$\{q_{F_1}(v_{21}, \ldots, v_{2g_2}, x_{g_2+1} \cdots x_{g_1}, z_1, z_2) \wedge$$
$$[Q_{F_1}(v_{21}, \ldots, v_{2g_2}, x_{g_2+1} \cdots x_{g_1}, z_1, z_2, v_{31}, \ldots, v_{3g_1}, z_3) \supset$$
$$\{q_{F_2}(v_{31}, \ldots, v_{3g_2}, x_k) \wedge$$
$$[Q_{F_2}(v_{31}, \ldots, v_{3g_2}, x_k, v_{41} \cdots v_{4g_2}, z_4) \supset$$
$$\phi_{\sigma_2}(y_1, \ldots, y_{\sigma_2'}, v_{41}, \ldots, v_{4g_2}, v_{3g_2+1} \cdots v_{3g_1}, x_{g_1+1}, \cdots$$
$$x_{j-1}, f_1(z_3, z_4), \ldots, x_{\sigma_2''})]]]]]]]] .$$

The construction is lengthy but the ideas behind it are quite simple.

b) The construction of W_P is unaffected.

c) The construction of W_{F_i} is as follows: if σ_0 is the first statement on the body of F_i, and σ_1 (i.e., null) is the last statement, then

$$W_{F_i} :- \ \{q_{F_i}(y_1 \cdots y_{g_i+n_i}) \supset$$
$$\phi_{\sigma_0}(y_1 \cdots y_{g_i+n_i}, y_1 \cdots y_{g_i}, \beta, y_{g_i+1} \cdots y_{g_i+n_i}, \beta \cdots \beta),$$
$$\phi_{\sigma_1}(y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \supset Q_{F_i}(y_1 \cdots y_{g_i+n_i}, x_1 \cdots x_{g_i+1})\} \ .$$

(Note that $g_i+n_i = \sigma_0' = \sigma_1'$.)

The reason why this construction of W_{F_i} works is that if a statement calls a procedure F_i, then the g_i globals of F_i

are exactly the first g_i variables for the statement in question (by the property of the ordering, mentioned previously).

6. Non-type procedures

We can now allow, as types of statement, procedures which do not return values. The effect of these 'non-type procedures' is purely on their global variables. The differences in programs are

i) We distinguish between the declarations of the two types of procedure by calling the non-type procedures routines. A routine declaration is simply

$$\underline{\text{routine}}\ R_i(X_{\beta_1}\ldots X_{\beta_{n_i}});$$
$$\langle\text{block}\rangle;$$

ii) In the body of a routine declaration, the name of the routine cannot be used like a variable. (This will require slight, but trivial, modifications of the definition of g_i .)

iii) Routines are called by statements of the form

$R_i(\tau_1\ldots\tau_{n_i})$, where $\tau_1\ldots\tau_{n_i}$ are expressions.

The modifications of the axioms are minor. As mentioned already, a slight but obvious modification is needed in the definition of g_i , so that routine names are not counted as variables in the way the procedure names are. Apart from this we need only consider the routine statements themselves and the routine declarations:

a) We shall consider only a very elementary routine statement. The treatment when more complicated expressions are included is similar

(but more complicated).

σ:- $R_i(X_{\alpha_1} \ldots X_{\alpha_{n_i}})$ and the successor of σ is σ_2

$$W_\sigma :- \phi_\sigma(y_1 \ldots y_{\sigma'}, x_1 \ldots x_{\sigma''}) \supset$$
$$\{q_{R_i}(x_1 \ldots x_{g_i}, x_{\alpha_1} \ldots x_{\alpha_{n_i}}) \wedge$$
$$[Q_{R_i}(x_1 \ldots x_{g_i}, x_{\alpha_1} \ldots x_{\alpha_{n_i}}, v_1 \ldots v_{g_i}) \supset$$
$$\phi_{\sigma_2}(y_1 \ldots y_{\sigma'_2}, v_1 \ldots v_{g_i}, x_{g_i+1}, \ldots, x_{\sigma''_2})]\} .$$

b) If σ_0 and σ_1 are the first and last statements in the body of R_i then

$$W_{R_i} :- \{q_{R_i}(y_1 \ldots y_{g_i+n_i}) \supset$$
$$\phi_{\sigma_0}(y_1 \ldots y_{g_i+n_i}, y_1 \ldots y_{g_i+n_i}, \beta \ldots \beta),$$
$$\phi_{\sigma_1}(y_1 \ldots y_{\sigma'_1}, x_1 \ldots x_{\sigma''_1}) \supset Q_{R_i}(y_1 \ldots y_{g_i+n_i}, x_1 \ldots x_{g_i})\} .$$

7. Data Types

The addition of data types to the language is more than a simple modification of the computing system; it implies a partition of the data space $|\mathcal{D}|$, and the existence of certain 'type-conversion' functions which do not appear explicitly in programs. We will therefore start by considering $\mathcal{D}$.

$\mathcal{D}$ is suitable for basing a multi-typed language on if there are

i) special subsets $A_1, A_2, \ldots$ of $|\mathcal{D}|$, possibly infinite in number, and not necessarily disjoint. Equivalently there are unary

relations $\pi_1, \pi_2, \ldots$ on $|\mathcal{A}|$ characterizing these subsets; and they correspond to unary predicate symbols in the basis of $\mathcal{A}$ which we shall denote by $T_1, T_2 \ldots$.

ii) for each subset A_i , a function $\psi_i : |\mathcal{A}| \to A_i$, such that for all $\xi \in A_i$, $\psi_i(\xi) = \xi$. The corresponding function symbol in the basis of $\mathcal{A}$ we shall denote by h_i .

iii) a special element from each subset A_i , corresponding to a constant symbol in the basis of $\mathcal{A}$ which we shall denote by β_i .

This means that $\mathcal{A}$ is a model of the following axioms:

$$T_i(h_i(x))$$

$$T_i(x) \supset h_i(x) = x$$

$$T_i(\beta_i)$$

for all i such that $A_i \subseteq |\mathcal{A}|$.

The sets A_i will be identified with the various types of data manipulated by programs, e.g. integer, real, complex, etc. The special element from each set is the initial value assigned to variables of the corresponding type, and the function ψ_i is a type-conversion function. The latter will be applied during execution of programs to ensure that variables of a given type only get assigned values of that type, and that procedures get called only with appropriate types of arguments.

Other type conversions during the evaluation of expressions will occur automatically because the functions corresponding to function symbols in programs must be total on $|\mathcal{A}|$. For example, if the subset

A_N of $|\mathcal{I}|$ consists of the integers, the function corresponding to the function symbol + may perform addition on A_N, but must also be defined for arguments of other types. Hence in specifying its operation for other types of arguments we can introduce any type conversion rules we wish. We have therefore shifted the bulk of the type conversion into specification of $\mathcal{I}$, and need not consider it further.

The changes made in programs by the addition of types is as follows:

i) When variables are declared, their type must be specified. This is done using the predicate symbols T_i, by replacing declarations of the form

decl X,Y,Z;

by, for example

$\underline{T_1}$ X;

$\underline{T_4}$ Y,Z;

ii) The types of formal parameters and input variables must be declared similarly, and the type of result a procedure returns must be specified. e.g.

a) instead of

procedure $F_i(X_1,X_2,X_3)$;

⋮

we might have

$\underline{T_3}$ procedure $F_i(\underline{T_4}\ X_1,\underline{T_2}\ X_2,X_3)$

b) instead of

program $P(X_3,X_5,X_1)$;

we might have

$$\underline{\text{program}}\ P(\underline{T_1}\ X_3, \underline{T_3}\ X_5, \underline{T_4}\ X_1);$$

and c) instead of

$$\underline{\text{routine}}\ R_i(X_1, X_2, X_{10});$$

we might have

$$\underline{\text{procedure}}\ R_i(\underline{T_1}\ X_1, \underline{T_4}\ X_2, \underline{T_3}\ X_{10});$$

(i.e., to conform with Algol notation we revert to using the word procedure for routines, now that the declarations indicate no value is returned.)

If $T_1, T_2, \ldots$ are in fact 'integer', 'real' ... the similarity to Algol is apparent.

The modifications of the axioms $\hat{W}_P$ are again very simple.

The axioms for a given program are almost those that would be produced if the program were changed slightly as follows.

i) Assignment statement

$$X_j := \tau$$

where X_j is declared of type T_i, is changed to

$$X_j := h_i(\tau) \ .$$

ii) Procedure or routine call

$$F_i(\tau_1 \ldots \tau_{n_i})$$

where the formal parameters of F_i are declared of types $T_{\alpha_1}, \ldots, T_{\alpha_{n_i}}$, is changed to

$$F_i(h_{\alpha_1}(\tau_1), \ldots, h_{\alpha_{n_i}}(\tau_{n_i})) \ .$$

The only other modifications of the axioms involves assigning the correct types of initial values.

e.g. for $\underline{T_3}$ $\underline{\text{procedure}}$ $F_i(\underline{T_2}\ X_1, \underline{T_3}\ X_2)$;

$\underline{\text{begin}}$

$\underline{T_1}\ X_4, X_5$;

$\underline{T_2}\ X_6$;

σ_0;

$\vdots$

$\underline{\text{null}}$

$\underline{\text{end}}$

W_{F_i} includes the axiom

$$q_{F_i}(y_1, \cdots, y_{g_i+n_i}) \supset$$

$$\phi_{\sigma_0}(y_1, \cdots, y_{g_i+n_i}, y_1, \cdots, y_{g_i}, \beta_3, y_{g_i+1} \cdots y_{g_i+n_i}, \beta_1, \beta_1, \beta_2) ,$$

and similarly for routines and blocks.

8. Boolean Procedures

In our treatment of data types we have been completely general, and assumed no properties of the types of data considered. However, there are some type of data which influence the syntax of programs, for example, 'boolean' data. Boolean variables are considered to hold 'truth values', and it is desirable to allow statements of the form

$$X_i := \pi$$

where X_i is a boolean variable and π is a propositional expression.

Using such variables it is possible to declare procedures returning values of type 'boolean', and it is desirable to allow such boolean procedures to be used in place of predicates in propositional expressions.

Clearly the values of boolean variables are not really truth and falsity, but some values that can be interpreted as truth and falsity. Therefore if $|\mathcal{I}|$ includes a set of boolean values, e.g. A_0, then there is a special relation χ characterizing the 'true' elements of A_0. The predicate symbol corresponding to this relation we shall denote by G. We shall also assume two special elements of A_0, corresponding to constants $\mathcal{T}$ and $\mathcal{F}$, only the first of which satisfies χ. That is, $\mathcal{I}$ satisfies the axiom

$$T_0(\mathcal{T}) \wedge T_0(\mathcal{F}) \wedge G(\mathcal{T}) \wedge \neg G(\mathcal{F}) \quad .$$

If there is such a data type in $\mathcal{I}$, then we can use boolean variables and boolean procedures in the way mentioned, and also use the constants $\mathcal{T}$ and $\mathcal{F}$ and any boolean variable as propositional expressions (in assignments to boolean variables, and in other propositional expressions).

The axioms $\hat{W}_P$ are then exactly as they would be if the program were changed as follows.

i) Wherever an <u>expression</u> τ is used where a <u>propositional expression</u> is appropriate (using the original definitions of expressions and propositional expressions) τ is replaced by $G(\tau)$.

ii) Wherever a <u>propositional expression</u> π is used where an <u>expression</u> is appropriate, π is replaced by <u>if</u> π <u>then</u> $\mathcal{T}$ <u>else</u> $\mathcal{F}$.

The program then conforms with the original syntax, and $\hat{W}_P$ can be constructed.

9. Arrays

Another type of data that influences the syntax of programs is arrays. If a program contains a variable X_i whose value is an n-dimensional array, then we would like to allow expressions of the form $X_i[\tau_1,\ldots,\tau_n]$, and statements of the form

$$X_i[\tau_1,\ldots,\tau_n] := \tau_{n+1} \ ,$$

where $\tau_1 \ldots \tau_{n+1}$ are expressions. We shall allow such expressions and statements by showing that they are simply shorthand forms of other expressions and statements of types we have already considered.

The axioms $\hat{W}_P$ are then constructed from the program with these other expressions and statements replacing the array constructions.

To represent these constructions as normal expressions and statements we require the arrays to be objects defined as follows.

We consider an n-dimensional array to be a family of data objects indexed by certain n-tuples of other objects (for generality we do not assume the index-objects are integers). The simplest situation is where the elements of the n-tuples are independent, i.e., the index set is the cartesian product of n sets of data objects. Only this situation will be considered here. Thus an n-dimensional array Ξ_n with index set $B_1 \times B_2 \times \ldots \times B_n$ consists of a family of objects $\{\xi_{b_{\alpha_1}, b_{\alpha_2} \ldots b_{\alpha_n}}\}$, $b_{\alpha_i} \epsilon B_i$, $i = 1 \ldots n$.

In normal Algol programs, $B_1 \ldots B_n$ are finite sets (of integers) that are determined during the computation of the program (prior to allocation of the array variable). If subsequently there is an attempt to use an index not from $B_1 \times \ldots \times B_n$ then either the program fails, e.g. for exceeding array bounds, or perhaps some conversion is made to

give an index within bounds, e.g. when a real number is used as an index. If we wish to keep $B_1 \ldots B_n$ as general sets of data objects, it is difficult to incorporate the feature of array bounds into the language, and even harder to give a logical definition.

We are therefore going to assume that the sets $B_1 \ldots B_n$ are sets of data types, e.g. $A_{\gamma_1} \ldots A_{\gamma_n}$. Then if there is an attempt to use an index not from $A_{\gamma_1} \times A_{\gamma_2} \times \ldots \times A_{\gamma_n}$, it is converted by using the functions $\psi_{\gamma_1} \ldots \psi_{\gamma_n}$. Then, for example, an integer-index array has no bounds on the integers that can be used as indices.

We are also going to assume that all the objects in a given array are of one type. An array variable declaration is therefore of the form

$$\underline{T_i \text{ array}}\ X_1[\underline{T_k}, \underline{T_h}, \underline{T_g}]$$

indicating the variable X_1 is to take as values families of objects from A_i, and all the families will have index set $A_k \times A_h \times A_g$.

Since arrays are data objects, we might consider the set of all arrays to form a data type A_α, with predicate symbol T_α. But then T_α could be used in array declarations,

e.g. $\underline{T_i \text{ array}}\ X_2[T_\alpha]$,

so that one index of X_2 would be X_2 itself. Such circularity will lead to paradoxes. It can be removed by letting there be many types of array. For instance, the type of X_1 above we could denote by $T_{(k,h,g,i)}$. This type could then be used in other array declarations, and no circularity would result.

e.g. $\underline{T_{(i,j)} \text{ array}}\ X_3[\underline{T_j}, \underline{T_{(k,h,g,i)}}]$,

X_3 would be of type $T_{(j,(k,h,g,i),(i,j))}$, namely a two-dimensional array

(of one-dimensional arrays) indexed by elements of A_j and by three-dimensional arrays of elements of A_i, indexed by elements of A_k, A_h and A_g.

Such objects are difficult to implement or understand*/ and in practice we would restrict ourselves to simple arrays, neither indexed by nor containing other arrays. However, we shall continue for a while to consider the general case.

For array objects to be used in programs as the values of array variables, $\mathcal{J}$ must contain two functions

$$\kappa_{(\gamma_1\cdots\gamma_i)}: A_{\gamma_1} \times \cdots \times A_{\gamma_{i-1}} \times A_{(\gamma_1\cdots\gamma_i)} \to A_{\gamma_i}$$

and

$$\alpha_{(\gamma_1\cdots\gamma_i)}: A_{\gamma_1} \times \cdots \times A_{\gamma_i} \times A_{(\gamma_1\cdots\gamma_i)} \to A_{(\gamma_1\cdots\gamma_i)}$$

for each set $A_{(\gamma_1\cdots\gamma_i)}$, corresponding to function symbols $c_{(\gamma_1\cdots\gamma_i)}$ and $a_{(\gamma_1\cdots\gamma_i)}$.

These functions are defined as follows: for $b_j \in A_{\gamma_j}$, $j = 1\ldots i$, and $\Xi \in A_{(\gamma_1\cdots\gamma_i)}$

$$\kappa_{(\gamma_1\cdots\gamma_i)}(b_1,\ldots,b_{i-1},\Xi) = \Xi_{b_1\cdots b_{i-1}}$$

(i.e., the element of Ξ indexed by $(b_1,\ldots,b_{i-1})$) and

$$\alpha_{(\gamma_1\cdots\gamma_i)}(b_1,\ldots,b_i,\Xi) = \Xi'$$

---

*/ although the type hierarchy of arrays is intriguingly similar to the type hierarchy of computable functions of Scott [ ], especially considering arrays as functions from index sets into elements of the arrays.

where Ξ' is the family of objects (in A_{γ_i}) identical to Ξ except that element $\Xi'_{b_1 \cdots b_{i-1}}$ is b_i.

These are McCarthy's [12] state-vector functions, generalized to an arbitrary number of dimensions of indices of arbitrary type. Using these functions we can clearly express the constructs

$$X_j[\tau_1 \cdots \tau_n]$$

and

$$X_j[\tau_1 \cdots \tau_n] := \tau_{n+1}$$

as follows. For array variable X_j, declared of type $T_{(\gamma_1 \cdots \gamma_{n+1})}$,

$$X_j[\tau_1 \cdots \tau_n]$$

in some expression is shorthand for

$$c_{(\gamma_1 \cdots \gamma_{n+1})}(h_{\gamma_1}(\tau_1), \ldots, h_{\gamma_n}(\tau_n), X_j)$$

and statement

$$X_j[\tau_1, \ldots, \tau_n] := \tau_{n+1}$$

is shorthand for

$$X_j := a_{(\gamma_1 \cdots \gamma_{n+1})}(h_{\gamma_1}(\tau_1), \ldots, h_{\gamma_{n+1}}(\tau_{n+1}), X_j) \quad .$$

Hence for any program P we can construct $\hat{W}_P$ using these function symbols $c_{(\gamma_1 \cdots \gamma_i)}$ and $a_{(\gamma_1 \cdots \gamma_i)}$.

Once again we have defined a construct in the language by shifting most of the task into the specification of $\mathcal{I}$. Now if $\mathcal{I}$ has certain simple properties, and programs are restricted in certain ways it is possible to give complete axioms Γ for the $\alpha_{(\gamma_1 \cdots \gamma_i)}$ and $\kappa_{(\gamma_1 \cdots \gamma_i)}$ functions, so that the task of specifying $\mathcal{I}$ simply involves finding a model of these axioms Γ. Better still, if $\mathcal{I}_o$ is $\mathcal{I}$ restricted to the non-array data types, then we can $\mathcal{I}_o$-relatively deduce from

$W_P(k,Q_P),\Gamma$ exactly the situations we can $\mathcal{I}$-relatively deduce from $W_P(k,Q_P)$. So adding arrays to the language based on $\mathcal{I}_o$ simply means that Γ is added to the logical definition, and programs are converted to use the $c_{\gamma_1\cdots\gamma_i}$ and $a_{\gamma_1\cdots\gamma_i}$ functions explicitly.

The restrictions on $\mathcal{I}$ are:

i) The family of objects in A_{γ_i} , comprising the initial value $\beta_{(\gamma_1\cdots\gamma_i)}$ of array variables of type $T_{(\gamma_1\cdots\gamma_i)}$, has all its elements equal to β_{γ_i} .

ii) The objects denoted by the constant symbols $b_1\ldots$ cannot be arrays.

iii) The only functions in $\mathcal{I}$ mapping into arrays are the type change functions $\psi_{(\gamma_1\cdots\gamma_i)}$ and the functions $\alpha_{(\gamma_1\cdots\gamma_i)}$ and $\kappa_{(\gamma_1\cdots\gamma_i)}$.

The restrictions ii) and iii) on $\mathcal{I}$ imply that the only expressions in programs that can have arrays as values are

a) X_j where X_j is an array variable,

b) $F_i(\tau_1\ldots\tau_{n_i})$ where F_i is an array procedure,

and c) $X_k[\tau_1\ldots\tau_n]$ where X_k is an array array;

and in all cases the type of array produced is known from the declarations. It is therefore possible to require the following restrictions on programs:

iv) In expressions of the form

$$f_i(\tau_1\ldots\tau_n) ,$$

none of the expressions $\tau_1\ldots\tau_n$ are array valued.

v) In expressions of the form

$$F_i(\tau_1\ldots\tau_{n_i}) ,$$

the j-th formal parameter of F_i is declared to be an array of a certain type if and only if the expression τ_j has as value an array of this type.

vi) In statements of the form

$$X_j := \tau$$

τ is an expression whose value is an array of a certain type if and only if X_j is declared to be an array variable of this type.

vii) No array variable can be declared to have arrays as either indices or elements, i.e., no higher type arrays.

viii) Input variables cannot be arrays.

ix) No predicate symbol in a program is applied to an expression that has an array as value.

These restrictions ensure that in the computations of programs, no type-conversion is performed from or into arrays. Therefore the functions $h_{(\gamma_1 \cdots \gamma_i)}$ need not be introduced when constructing $\hat{W}_\sigma$. Then, in the symbolic computation performed by $\mathfrak{A}_k^\xi$-relative deduction from $W_P(k,Q_P)$, the terms representing array and non-array data objects will be of certain forms, as defined below.

A <u>simple-type term</u> is one of the following forms

i) $\left.\begin{matrix} b_i \\ \beta_i \end{matrix}\right\}$ simple type constants

ii) k_i input value

iii) $\begin{matrix} f_i(\tau_1 \ldots \tau_n) \\ h_i(\tau_1 \ldots \tau_n) \end{matrix}$ where $\tau_1 \ldots \tau_n$ are simple-type terms.

iv) $c_{(\gamma_1 \ldots \gamma_i)}(h_{\gamma_1}(\tau_1), \ldots, h_{\gamma_{i-1}}(\tau_{i-1}), \tau)$

where $\tau_1 \ldots \tau_{i-1}$ are simple-type terms and τ is a $T_{(\gamma_1 \ldots \gamma_i)}$-term.

Simple-type terms denote non-array objects. A simple-type term that does not contain $c_{(\gamma_1 \ldots \gamma_i)}$ symbols is called a non-array term.

A $T_{(\gamma_1 \ldots \gamma_i)}$-term is one of the following forms:

i) $\beta_{(\gamma_1 \ldots \gamma_i)}$ array type constant

ii) $a_{(\gamma_1 \ldots \gamma_i)}(h_{\gamma_1}(\tau_1), \ldots, h_{\gamma_i}(\tau_i), \tau)$

where $\tau_1 \ldots \tau_i$ are simple-type terms and τ is a $T_{(\gamma_1 \ldots \gamma_i)}$-term.

$T_{(\gamma_1 \ldots \gamma_i)}$-terms denote array objects of type $T_{(\gamma_1 \ldots \gamma_i)}$.

Restriction ix) implies that in $\mathcal{J}_k^{\xi}$-relative deduction from $W_P(k, Q_P)$ we need only know the truth or falsity of predicates applied to simple-type terms. We shall give a set of axioms Γ such that for every simple type term τ we can ($\mathcal{J}_{o\,k}^{\xi}$-relatively) deduce from Γ a formula $\tau = \tau'$ where τ' is some non-array term denoting the same value as τ. Since non-array terms are constructed from the basis symbols of $\mathcal{J}_o$ ($\mathcal{J}$ restricted to non-array data objects), then we can deduce all situations from $W_P(k, Q_P), \Gamma$ relative to $\mathcal{J}_{o\,k}^{\xi}$.

Γ is therefore the required set of axioms describing the addition of arrays to the language based on $\mathcal{J}_o$.

The axioms Γ are: for all $\gamma_1 \ldots \gamma_i$ s.t. $A_{\gamma_1}, \ldots, A_{\gamma_i} \subseteq |\mathcal{A}|$,

I: $\tau_1 \neq \tau_1' \vee \ldots \vee \tau_{i-1} \neq \tau_{i-1}' \supset$

$$c_{(\gamma_1 \ldots \gamma_i)}(\tau_1, \ldots, \tau_{i-1}, a_{(\gamma_1 \ldots \gamma_i)}(\tau_1', \ldots, \tau_{i-1}', \tau_i, \tau)) = c_{(\gamma_1 \ldots \gamma_i)}(\tau_1, \ldots, \tau_{i-1}, \tau) \quad .$$

II: $c_{(\gamma_1 \ldots \gamma_i)}(\tau_1, \ldots, \tau_{i-1}, a_{(\gamma_1 \ldots \gamma_i)}(\tau_1, \ldots, \tau_{i-1}, \tau_i, \tau)) = \tau_i$

III: $c_{(\gamma_1 \ldots \gamma_i)}(\tau_1, \ldots, \tau_{i-1}, \beta_{(\gamma_1 \ldots \gamma_i)}) = \beta_{\gamma_i}$

where $\tau_1, \tau_1', \tau_2, \tau_2', \ldots, \tau_i$ are simple-type terms and τ is a $T_{(\gamma_1 \ldots \gamma_i)}$-term.

These axioms are simply generalizations of those for McCarthy's c and a functions.

It is easy to prove the following property of Γ by induction on τ .

Completeness of Γ :

For all simple-type terms τ , and non-array terms τ' :

$$\vdash_{\Gamma}^{\mathcal{A}_o\, {}^{\xi}_{k}} \tau = \tau' \Leftrightarrow \mathcal{A}_k^{\xi}(\tau) = \mathcal{A}_o\, {}^{\xi}_{k}(\tau') \quad .$$

Hence Γ can be used in the desired way to add arrays to structure $\mathcal{A}_o$.

10. Jumps out of procedure bodies

This final extension of the Simple Language is also the most difficult to incorporate into the logical definition.

We allow the labels in jump statements within procedure bodies to refer to statements in blocks enclosing the procedure declarations. The variables in scope for these latter statements will be included in the globals of the jump statements. In fact, if statement σ is the destination of jump statement σ_1, then the variables in scope for σ will be exactly the first σ'' of the σ_1'' globals of σ_1.

Therefore, at first sight, it appears that the formula W_{σ_1}, for the jump statement,

$$\text{i.e.,} \quad \phi_{\sigma_1}(y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \supset \phi_{\sigma}(y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''})$$

will already adequately deal with such jumps out of procedures.

However, variables $y_1 \cdots y_{\sigma_1'}$ in the first part of W_{σ_1} represent the values of the globals and actual parameters when the smallest procedure enclosing σ_1 was last called (we shall call these the 'called-values' of σ_1). Since σ is not within this procedure, $y_1 \cdots y_{\sigma'}$ do not represent the called values for the subsequent computation of σ. Therefore W_{σ_1} does not follow the computation of σ_1 in the required way.

Now the called values of a statement do not affect the computation of the statement; they are only used in the logical definition where they are merely passed on from one statement to the next until the computation of the procedure is completed. In a similar way, W_{σ_1} could be modified to cope with jumps out of procedures if for each statement σ_i there were

another set of σ_i° values, taken from variable values at previous points in the computation, which we shall call the 'historical-values' of the statement. These values are to be passed on (by the formulae) from statement to statement until some jump statement σ_1 is reached. At this point we would require that the historical values contain the called values of destination σ. In fact, if the first σ' of the historical values of σ_1 are the called values of σ we might define W_{σ_1} to be

$$\phi_{\sigma_1}(w_1 \cdots w_{\sigma_1^\circ}, y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \supset \phi_\sigma(w_1 \cdots w_{\sigma'}, x_1 \cdots x_{\sigma''}) \ .$$

However, σ itself will have σ° historical values which W_{σ_1} must supply from the historical values of σ_1. So if

i) the first σ° historical values of σ_1 are the historical values of σ, and

ii) the next σ' historical values are the called values of σ

we can define W_{σ_1} to be

$$\phi_{\sigma_1}(w_1 \cdots w_{\sigma_1^\circ}, y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \supset$$
$$\phi_\sigma(w_1 \cdots w_{\sigma^\circ}, w_{\sigma^\circ+1}, \cdots, w_{\sigma^\circ+\sigma'}, x_1 \cdots x_{\sigma''}) \ .$$

In fact it is possible to find historical values with just the properties mentioned.

If the smallest procedure declaration enclosing a statement is for procedure F_i, then we say that the statement is at the top level of F_i. We stipulate that all statements at the top level of F_i have the same historical values (because we assume that from any such

statement it is possible to get to any of the jump statements that jump out of F_i).

With each procedure F_i we associate the set of statements $\{\Sigma_i\}$ which is the union of

i) The set of all statements outside F_i that are the destinations of statements within F_i .

ii) $\{\Sigma_j\}$ for all procedures F_j called within F_i that are not declared within F_i .

Now, if F_j is the smallest procedure containing a statement in $\{\Sigma_i\}$, then it immediately follows that the historical and called values of the statements at the top level of F_j contain just the historical and called values of statements in $\{\Sigma_i\}$ (any statement in $\{\Sigma_i\}$ outside F_j must be in $\{\Sigma_j\}$). That is, the historical values of statements at the top level of F_i are just the historical and called values of statements at the top level of F_j , where F_j is the smallest procedure containing a statement in $\{\Sigma_i\}$. (This relation between F_i and F_j will be denoted by $R(F_i,F_j)$.)

This is sufficient to specify the number σ° of historical values for any statement σ . In addition, it implies that the historical values of σ (at the top level of F_k) can be ordered as follows:

the historical values of statements at top level of F_h (in order) followed by the called values of statements at top level of F_h ,

where $R(F_k,F_h)$.

Then if σ_1 is in $\{\Sigma_k\}$, the first σ_1° historical values of σ are the historical values of σ_1 , and the next σ_1' historical values of σ are the called values of σ_1 .

This property of the ordering is just what we need to modify the logical definition.

We increase the argument places of the various predicate symbols as follows:

For statement σ, ϕ_σ is a $\sigma^\circ + \sigma' + \sigma''$ -ary predicate symbol. For procedure F_i, we define a number e_i so that $e_i = \sigma^\circ$ for any statement σ at the top level of F_i. Then q_{F_i} is a $e_i+g_i+n_i$ -ary predicate symbol. As before, Q_{F_i} is a $g_i+n_i+g_i+1$ -ary predicate symbol.

We then modify the formulae W_σ and W_{F_i} as follows:

i) If σ is a jump out of a procedure, to statement σ_1, then

$$W_\sigma :- \phi_\sigma(w_1 \cdots w_{\sigma^\circ}, y_1 \cdots y_{\sigma'}, x_1 \cdots x_{\sigma''}) \supset$$

$$\phi_{\sigma_1}(w_1 \cdots w_{\sigma_1^\circ}, w_{\sigma_1^\circ+1}, \cdots, w_{\sigma_1^\circ+\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \quad .$$

ii) For other statements, W_σ is modified by

a) adding variable symbols $w_1 \cdots w_{\sigma_i^\circ}$ onto all subformulae of the form

$$\phi_{\sigma_i}(t_1 \cdots t_{\sigma_i'}, \tau_1 \cdots \tau_{\sigma_i''})$$

giving

$$\phi_{\sigma_i}(w_1 \cdots w_{\sigma_i^\circ}, t_1 \cdots t_{\sigma_i'}, \tau_1 \cdots \tau_{\sigma_i''}) \quad ;$$

b) adding to all subformulae of the form

$$q_{F_j}(\tau_1 \cdots \tau_{g_j+n_j})$$

the first e_j variable symbols in the list

$$w_1 \cdots w_{\sigma^\circ} y_1 \cdots y_{\sigma'},$$

giving either

$$q_{F_j}(w_1 \cdots w_{e_j}, \tau_1 \cdots \tau_{g_j+n_j})$$

or

$$q_{F_j}(w_1 \cdots w_{\sigma^\circ}, y_1 \cdots y_{\sigma'}, \tau_1 \cdots \tau_{g_j+n_j}) \quad .*/$$

iii) If σ_0 and σ_1 (i.e., null) are the first and last statements in F_i, then

$$W_{F_i} :- \{q_{F_i}(w_1 \cdots w_{e_i}, y_1 \cdots y_{g_i+n_i}) \supset \phi_{\sigma_0}(w_1 \cdots w_{\sigma_0^\circ}, y_1 \cdots y_{g_i+n_i}, y_1 \cdots y_{g_i}, \beta, y_{g_i+1}, \ldots, y_{g_i+n_i}, \beta \cdots \beta),$$

$$\phi_{\sigma_1}(w_1 \cdots w_{\sigma_1^\circ}, y_1 \cdots y_{\sigma_1'}, x_1 \cdots x_{\sigma_1''}) \supset Q_{F_i}(y_1 \cdots y_{g_i+n_i}, x_1 \cdots x_{g_i+1})\}$$

$$(e_i = \sigma_0^\circ = \sigma_1^\circ \quad \text{and} \quad g_i+n_i = \sigma_0' = \sigma_1' \ .)$$

The construction of $\hat{W}_P$ should become clear from the following example, (for simplicity we have dropped the data types).

---

*/ If σ is at the top level of the smallest procedure enclosing a statement in $\{\Sigma_j\}$ then $e_j = \sigma^\circ + \sigma'$, otherwise $e_j \leq \sigma_i^\circ$.

```
program P(X1,X2,X3);
   begin
      decl X4, X5;
      procedure F6(X7,X8);
         begin
            decl X9;
            procedure F10(X11,X12);
               begin
                  σ1;
                  ⋮
      σ2 -------- F10 := F6(X7,X12);
      σ3 -------- goto L;
                  ⋮
      σ4 -------- goto M;
                  ⋮
      σ5 -------- null
               end;
            σ6;
            ⋮
      σ7 - L:F6 := F10(X4,X3);
      σ8 -- goto N;
            ⋮
      σ9 -- null
         end;
      σ10;
      ⋮
σ11 --- M: X2 := F6(X4,X2);
      N: σ12;
      ⋮
σ13 ---  output(X4,X2)
   end
```

Here

$$\{\Sigma_6\} = \{\sigma_{11}, \sigma_{12}\}$$

$$\{\Sigma_{10}\} = \{\sigma_7, \sigma_{11}\} \cup \{\Sigma_6\} = \{\sigma_7, \sigma_{11}, \sigma_{12}\} \ .$$

Hence

$$\sigma^\circ_{10} = \sigma^\circ_{11} = \sigma^\circ_{12} = \sigma^\circ_{13} = 0 \ ;$$

$$e_6 = \sigma^\circ_6 = \sigma^\circ_7 = \sigma^\circ_8 = \sigma^\circ_9 = 3; \quad \sigma'_6 = \sigma'_7 = \sigma'_8 = \sigma'_9 = 7$$

$$e_{10} = \sigma^\circ_1 = \sigma^\circ_2 = \sigma^\circ_3 = \sigma^\circ_4 = \sigma^\circ_5 = 3+7 = 10 \ .$$

The various formulae are as follows:

$$W_P\text{:-} \quad \{q_P(y_1,y_2,y_3) \supset \phi_{\sigma_{10}}(y_1,y_2,y_3,y_1,y_2,y_3,\beta,\beta),$$
$$\phi_{\sigma_{13}}(y_1,y_2,y_3,x_1,\ldots,x_5) \supset Q_P(y_1,y_2,y_3,x_4,x_2)\}$$

$$W_{\sigma_{11}}\text{:-} \quad \phi_{\sigma_{11}}(y_1,y_2,y_3,x_1,\ldots,x_5) \supset$$
$$\{q_{F_6}(y_1,y_2,y_3,x_1,\ldots,x_5,x_4,x_2) \wedge$$
$$[Q_{F_6}(x_1,\ldots,x_5,x_4,x_2,v_1,\ldots,v_5,z_1) \supset$$
$$\phi_{\sigma_{12}}(y_1,y_2,y_3,v_1,z_1,v_3,v_4,v_5)]\}$$

$$W_{F_6}\text{:-} \quad \{q_{F_6}(w_1,w_2,w_3,y_1,\ldots,y_7) \supset$$
$$\phi_{\sigma_6}(w_1,w_2,w_3,y_1,\ldots,y_7,y_1,\ldots,y_5,\beta,y_6,y_7,\beta) \ ,$$
$$\phi_{\sigma_9}(w_1,w_2,w_3,y_1,\ldots,y_7,x_1,\ldots,x_9) \supset$$
$$Q_{F_6}(y_1,\ldots,y_7,x_1\ldots x_6)\}$$

$$W_{\sigma_7} :- \phi_{\sigma_7}(w_1,w_2,w_3,y_1,\ldots,y_7,x_1,\ldots,x_9) \supset$$
$$\{q_{F_{10}}(w_1,w_2,w_3,y_1,\ldots,y_7,x_1,\ldots,x_9,x_4,x_3) \wedge$$
$$[Q_{F_{10}}(x_1,\ldots,x_9,x_4,x_3,v_1\ldots v_9,z_1) \supset$$
$$\phi_{\sigma_8}(w_1,w_2,w_3,y_1,\ldots,y_7,v_1,\ldots,v_5,z_1,\ldots,v_9)]\}$$

$$W_{\sigma_8} :- \phi_{\sigma_8}(w_1,w_2,w_3,y_1,\ldots,y_7,x_1\ldots x_9) \supset \phi_{\sigma_{12}}(w_1,w_2,w_3,x_1,\ldots,x_5)$$

$$W_{F_{10}} :- \{q_{F_{10}}(w_1,\ldots,w_{10},y_1,\ldots,y_{11}) \supset$$
$$\phi_{\sigma_1}(w_1,\ldots,w_{10},y_1,\ldots,y_{11},y_1,\ldots,y_9,\beta,y_{10},y_{11}),$$
$$\phi_{\sigma_5}(w_1\ldots w_{10},y_1,\ldots,y_{11},x_1,\ldots,x_{12}) \supset$$
$$Q_{F_{10}}(y_1,\ldots,y_{11},x_1,\ldots,x_{10})\}$$

$$W_{\sigma_2} :- \phi_{\sigma_2}(w_1,\ldots,w_{10},y_1,\ldots,y_{11},x_1,\ldots,x_{12}) \supset$$
$$\{q_{F_6}(w_1,w_2,w_3,x_1,\ldots,x_5,x_7,x_{12}) \wedge$$
$$[Q_{F_6}(x_1,\ldots,x_5,x_7,x_{12},v_1,\ldots,v_5,z_1) \supset$$
$$\phi_{\sigma_3}(w_1,\ldots,w_{10},y_1,\ldots,y_{11},v_1,\ldots,v_5,x_6\ldots x_9,z_1,x_{11},x_{12})]\}$$

$$W_{\sigma_3} :- \phi_{\sigma_3}(w_1,\ldots,w_{10},y_1,\ldots,y_{11},x_1,\ldots,x_{12}) \supset$$
$$\phi_{\sigma_7}(w_1,w_2,w_3,w_4,\ldots,w_{10},x_1,\ldots,x_9)$$

$$W_{\sigma_4} :- \phi_{\sigma_4}(w_1,\ldots,w_{10},y_1\ldots y_{11},x_1,\ldots,x_{12}) \supset$$
$$\phi_{\sigma_{11}}(w_1,w_2,w_3,x_1,\ldots,x_5) \quad .$$

Conclusion

We have now developed a logical definition of a language with many of the features of Algol. This logical definition can also be used as a formalization of partial correctness, and therefore it can be used to formalize many properties of Algol programs. The definition has not yet been put to this use for any programs of interest, and it remains to be seen whether it can be used in practice for this purpose. The axioms $W_P(k,Q_P)$ produced for a program of any complexity will be numerous and complicated, but if the program itself forms self-contained sections, (e.g. procedures) then the axioms also will form self-contained groups, which can be used to prove properties of the sections. With practice, the definition could become a useful tool for practical program verification.

Acknowledgments

I am indebted to Dr. J. J. Florentin, of Imperial College, for introducing me to the subject of language definition, and for supervising the work presented here. His interest and encouragement were invaluable.

I would also like to thank Mr. C. D. Allen, of IBM Hursley, for many stimulating discussions.

Finally, I would like to thank Mrs. Phyllis Winkler, for magnificently accomplishing the unenviable task of typing this thesis.

References

[1] E. A. Ashcroft. 'Functional Programs as Axiomatic Theories.' C.C.A. Report No. 9. Centre for Computing and Automation, Imperial College, London.

[2] E. A. Ashcroft and Z. Manna. 'Formalization of Properties of Parallel Programs.' Artificial Intelligence Memo 110. Stanford University.

[3] R. M. Burstall. 'Formal description of program structure and semantics in first order logic.' Machine Intelligence 5. Edinburgh University Press. 1969.

[4] R. W. Floyd. 'Assigning Meaning to Programs.' Proc. of Symposia in Applied Math., Am. Math. Soc, 19 (1967), 19-32.

[5] P. J. Landin. 'A Correspondence Between ALGOL and Church's Lambda-Notation.' Comm. A.C.M., Vol. 8. Feb-March 1965.

[6] P. Lucas, et al. 'Informal Introduction to the Abstract Syntax and Interpretation of PL/I.' IBM Technical Report TR.25.083.

[7] Z. Manna. 'Termination of Algorithms.' Ph.D. Thesis. Carnegie Mellon University. Pittsburgh.

[8] _____ 'The Correctness of Programs.' Journal of Computer and System Sciences, Vol. 3. May 1969.

[9] _____ 'The Correctness of Non-deterministic Programs.' Artificial Intelligence Journal. Vol. 1, No. 1.

[10] _____ 'Mathematical Theory of Partial Correctness.' To appear in Symposium on the Semantics of Algorithmic Languages. (E. Engeler, Ed.) Springer Verlag. 1970.

[11] _____ and A. Pnueli. 'Formalization of Properties of Functional Programs.' To appear in J.A.C.M. (July 1970)

[12] J. McCarthy. 'A Formal Description of a Subset of Algol.' Formal Language Description Languages for Computer Programming. 1966.

[13] _____ and J. A. Painter. 'Correctness of a Compiler for Arithmetic Expressions.' Proc. of a Symposium in Applied Math. Vol. 19 - Math. Aspects of Computer Science. 1967.

[14] J. A. Painter. 'Semantic Correctness of a Compiler for an Algol-like Language.' Ph.D. Thesis. Artificial Intelligence Memo 44. Stanford University.

[15] D. Park. 'Fixpoint Induction and Proofs of Program Properties.' Machine Intelligence 5. Edinburgh University Press. 1969.

[16] J. R. Schoenfield. Mathematical Logic. Addison-Wesley Co. 1967.

[17] D. Scott. 'A Type-theoretical Alternative to CUCH, ISWIM, OWHY.' Unpublished paper.

[18] _____ 'Models of the X-calculus.' Unpublished paper.

[19] A. Tarski. 'A Lattice-theoretical Fixpoint Theorem and its Applications.' Pacific Journal of Maths. 5. 285-309.