SEMIAUTOMATIC SYNTHESIS OF INDUCTIVE PREDICATES

by

Mark Moriconi

June 1974 ATP-16

Semiautomatic Synthesis of Inductive Predicates

Mark Moriconi*

The University of Texas at Austin

1. Introduction.

The assertion approach for verifying programs was formalized by Floyd [9] and Naur [18]. Previous work in the field has indicated various practical limitations. It has been pointed out by King [13], Elspas, et al. [8], and Elspas, Levitt, and Waldinger [7] among others that one of the single most important factors limiting these efforts at program verification is the difficulty of inventing Floyd assertions. The difficulty appears to be not so much a problem of syntax or the assertion language, but one of correctly understanding the program.

Previous attempts at semiautomatic generation of inductive assertions exhibit essentially two approaches. The first is basically heuristic and has been suggested by Cooper [5], Katz and Manna [12], Wegbreit [22], and others. The second attempts to be somewhat more formal having as its basis a suggestion by Green [10] that the problem can be viewed as that of finding the solution to simultaneous sets of difference equations.

Our approach for finding a practical solution to the problem is significantly different from most others. We begin by formally developing a new representation of a program (We assume the program has an input assertion specified a priori.) which turns out to be an extremely useful tool in clarifying what a program (or part of a program) computes. This representation is called a case description and is not totally unlike the informal ideas presented by Pratt in [19,20]. A case description is a functional

---

*This work is based in part on the author's Ph.D. Thesis [17]. The work was supported by NSF Grant GJ-32269.

representation of a program consisting of a set of descriptors which are composed of control and kernel sets. The kernel set of a descriptor specifies the outputs which are computed when the control set is satisfied for a given input assignment. The functional equivalence of a program schemata and its corresponding case description for a given interpretation and initial assignment is proved in Section 5. As a result we view the abstract program and its associated case description as simply different representations of the same (partial) function.

We then proceed to develop various heuristics which when used in concert with a (partial) case description for a particular program generally yield consistent loop invariants without much difficulty. Examples of this are presented in Section 7.

2. Mathematical Background.

In this section we present a simple formulation of the first-order predicate calculus and develop concepts and notation to be used throughout. For further discussion see Church [4], Mendelson [16], and Schoenfield [21].

The basic alphabet consists of commas; parentheses; the logical symbols $\vee,\wedge,\sim$; individual variables $x_1, x_2, \ldots, x_n, \ldots$; individual constants $c_1, c_2, \ldots, c_n, \ldots$; function letters $f_1^1, f_1^2, \ldots, f_k^\ell, \ldots$; and predicate letters $p_1^1, p_1^2, \ldots, p_k^\ell, \ldots$. The superscript of a function or predicate letter represents the number of arguments, whereas the subscript is simply an index number to distinguish different function or predicate letters with the same number of arguments.

Terms are generated as follows:

1. Variables and individual constants are terms.
2. If $t_1, \ldots, t_n$ are terms, then $f_i^n(t_1, \ldots, t_n)$ is a term.

Atomic formulas are formed by applying predicate letters to terms, i.e. if $t_1, \ldots, t_n$ are terms, then $p_i^n(t_1, \ldots, t_n)$ is an atomic formula.

Well-formed formulas (wffs) are defined as follows:

1. An atomic formula is a wff.
2. If $\mathcal{S}$ and $\mathcal{T}$ are wffs, and x_i is a variable, then $(\sim\mathcal{S})$, $(\mathcal{S}\wedge\mathcal{T})$, and $((\forall x_i)\mathcal{S})$ are wffs.

Parentheses are omitted whenever their omission causes no confusion.

The scope of a quantifier occurring in a wff is defined to be the wff to which the quantifier applies. An occurrence of a variable in a wff is bound iff this occurrence is within the scope of a quantifier employing this variable, or is the occurrence in that quantifier. Otherwise, an occurrence of a variable is said to be free. Clearly a variable can be both free and

bound in the same wff. A wff is said to be quantifier-free if it contains no quantifiers.

An interpretation[1] $\mathcal{I}$ consists of a non-empty set $\underline{D}^{\dagger}$, called the domain of $\mathcal{I}$, and an assignment to each predicate letter p_i^n an n-ary relation $\underline{R}_i^n$ in $\underline{D}$, to each function letter f_i^n an n-ary total function $\underline{F}_i^n$ from $\underline{D}^n$ into $\underline{D}$, and to each individual constant c_i some fixed element d_{c_i} of $\underline{D}$.

We now define the intuitive notions of satisfiability and truth. Given an interpretation $\mathcal{I}$ with domain $\underline{D}$, let $\sum$ be the set of denumerable sequences of elements of $\underline{D}$. We define what it means for a sequence $\bar{s} = (a_1, a_2, \ldots)$ in $\sum$ to satisfy a wff $\mathcal{A}$ under a given interpretation $\mathcal{I}$.

As a preliminary step we define a function ρ of one argument having terms as arguments and values in $\underline{D}$.[2] An assignment is determined by the function ρ depending on $\bar{s}$, denoted as $\rho_{\bar{s}}$, in the following manner:

1. $\rho_{\bar{s}}(x_i) = a_i$.
2. $\rho_{\bar{s}}(c_i) = d_{c_i}$.
3. $\rho_{\bar{s}}(f_i^n(t_1, \ldots, t_n)) = \underline{F}_i^n(\rho_{\bar{s}}(t_1), \ldots, \rho_{\bar{s}}(t_n))$.

We define inductively what it means for a sequence $\bar{s}$ in $\sum$ to satisfy a wff as follows:

1. $\bar{s}$ satisfies the atomic formula $p_i^n(t_1, \ldots, t_n)$ iff $(\rho_{\bar{s}}(t_1), \ldots, \rho_{\bar{s}}(t_n)) \in \underline{R}_i^n$.
2. $\bar{s}$ satisfies $\sim\mathcal{A}$ iff $\bar{s}$ does not satisfy $\mathcal{A}$.
3. $\bar{s}$ satisfies $\mathcal{A} \wedge \mathcal{B}$ iff $\bar{s}$ satisfies $\mathcal{A}$ and $\bar{s}$ satisfies $\mathcal{B}$.

---

† Individual capital letters which are underscored represent block letters.

1. In Shoenfield [3] what we refer to as an interpretation is called a structure. We use this terminology to be consistent with subsequent definitions.
2. The ρ function is used extensively in Section 3.3 and Section 5.

4. $\bar{s}$ satisfies $(\forall x_i)\mathcal{A}$ iff every sequence of Σ differing from $\bar{s}$ in at most the i-th component satisfies $\mathcal{A}$.

$\mathcal{A}$ is said to be <u>satisfiable</u> iff there is an interpretation $\mathcal{I}$ for which $\mathcal{A}$ is satisfied by at least one sequence in Σ.

$\mathcal{A}$ is said to be <u>true</u> iff every sequence in Σ satisfies $\mathcal{A}$.

To say that a wff $\mathcal{A}$ is <u>logically valid</u> means that $\mathcal{A}$ is true for every interpretation $\mathcal{I}$.[3]

---

3. We could alternatively (by Gödel's Completeness Theorem) have singled out the logically valid wffs by specifying axioms and rules of inference.

3. Definitions of Abstract Programs and Related Concepts.

In this section we present a simple abstract model of a computer program (called a program or abstract schemata) and then consider a particular computation (execution sequence) resulting from a specific interpretation and assignment. Models similar to ours have been studied by many authors, e. g., Kaplan [11], Luckam, Park, and Paterson [14], and Manna [15].

3.1 Abstract Programs. An abstract program AP consists of :

1. (a) A finite set of individual input variables $\bar{x} = \{x_1, \ldots, x_n\}$ with $n \geq 0$, and

   (b) a finite set of individual program variables $\bar{y} = \{y_1, \ldots, y_n\}$ with $n \geq 1$.

2. A finite, directed, labeled graph which we define by the triple $\langle \mathcal{N}, \Delta, \mathcal{L} \rangle$ with a finite set of nodes $\mathcal{N} = \mathcal{N}' \cup \mathcal{H}$ s.t. there is

   (a) a unique entry node called START with START $\in \mathcal{N}'$, and

   (b) a set $\mathcal{H}$ composed of at least one exit node called HALT with $\mathcal{H} \cap \mathcal{N}' = \emptyset$.

3. $\Delta: \mathcal{N}' \to \mathcal{N} \cup \bigtimes_{i=1}^{2} \mathcal{N}$ defines the flow of control for AP.

4. $\mathcal{L}: \mathcal{N}' \to \mathcal{A} \cup \mathcal{B}$ gives a labeling for the nodes of AP with statements from the sets $\mathcal{A}$ and $\mathcal{B}$ of assignment and branch statements, respectively, which are defined as follows:

   (a) An assignment statement is an expression of the form $\alpha \leftarrow \tau(\bar{x},\bar{y})$ where $\tau(\bar{x},\bar{y})$ is a term with no variables other than x_i and y_i and α is a variable. In addition START is considered to be in $\mathcal{A}$. A node $n \in \mathcal{N}'$ is an assignment node iff $\forall n \in \mathcal{N}'$, $\mathcal{L}(n) \in \mathcal{A} \iff \Delta(n) \in \mathcal{N}$.

   (b) A branch statement is a quantifier-free wff $\beta(\bar{x},\bar{y})$ with no variables other than x_i and y_i. A node $n \in \mathcal{N}'$ is a branch node iff $\forall n \in \mathcal{N}'$, $\mathcal{L}(n) \in \mathcal{B} \iff \Delta(n) \in \mathcal{N}^2$.

3.2 Programs. Let $D_{\bar{x}}$ and $D_{\bar{y}}$ be non-empty domains for $\bar{x}$ and $\bar{y}$, respectively, such that $D_{\bar{x}} \subseteq D_{\bar{y}}$. For convenience (although it is not

necessary) we begin the schemata (first node after START) with an initial assignment statement which establishes initial values for all program variables. We denote the initial assignment statement by $\bar{y} \leftarrow \overline{\tau(\bar{x})}$ which represents a sequence of assignments $y_1 \leftarrow \tau_1(\bar{x}), \ldots, y_\ell \leftarrow \tau_\ell(\bar{x})$ where $\tau_j(\bar{x})$, $1 \leq j \leq \ell$, is a term whose only variables are x_i. In addition we require that a program schemata have at least one input variable or at least one constant.

Given an abstract program with the above characteristics, an interpretation $\mathcal{I}$ for $\underline{AP}$ specifies the following:

1. For each n-ary function symbol f_i^n appearing in $\underline{AP}$ we assign a total function from $D_{\bar{y}}^n$ into $D_{\bar{y}}$.
2. For each n-ary predicate symbol p_i^n appearing in $\underline{AP}$ corresponds an n-ary relation R_i^n in $D_{\bar{y}}$.
3. For each constant we assign a fixed element of $D_{\bar{y}}$.

An abstract program $\underline{AP}$ together with an interpretation $\mathcal{I}$ forms what is called a program and is denoted by $(\underline{P},\mathcal{I})$.

We now extend $\underline{AP}$ by adding an input predicate (or input assertion), denoted $\varphi(\bar{x})$, which is a wff with no free individual variables other than $\bar{x}$ and denote it as $(\underline{AP},\varphi)$. $\varphi(\bar{x})$ usually specifies the domains of the input variables and any constraints on the joint occurrence of values of input variables. $(\underline{AP}, \varphi)$ together with an interpretation $\mathcal{I}$ forms $(\underline{P},\mathcal{I},\varphi)$ where $\varphi(\bar{x})$ is interpreted in the usual manner over $D_{\bar{x}}$.

3.3 Interpreted Programs. Let $(\underline{P},\mathcal{I},\varphi)$ be a program and $\bar{\xi} \in D_{\bar{x}}$ be an input assignment for $\bar{x}$. This defines the interpreted program $(\underline{P},\mathcal{I},\varphi,\bar{\xi})$. An interpreted program can be executed defining what is called an execution sequence which may be finite or infinite. Before defining this concept we make the following notational conveniences to be used throughout.

Notation. Let $\langle d \rangle$ denote the vector[4] of constants $\langle d_1,\ldots,d_n \rangle$ and $\langle t \rangle$ the vector of terms $\langle t_1,\ldots,t_n \rangle$.

Definition 1. We define $\mathscr{E}$ inductively by

$$\mathscr{E}(1) = (\langle t_1 \rangle, \mathscr{C}_1, n_1)$$

when

$$\bar{\xi} \text{ satisfies } \varphi(\bar{x}),$$

where

$$n_1 = \Delta(\underline{\text{START}}),$$

$$\mathscr{C}_1 = \{\varphi(\bar{x})\},$$

and

$$t_1 = x_\nu, \qquad \nu \text{ fixed},$$

if there are input variables, or

$$t_1 = c_\nu, \qquad \nu \text{ fixed},$$

if there are no input variables.

Henceforth, when it causes no confusion we omit references to $\bar{x}$ as its value remains constant under $\rho_{\bar{\xi}}$. We also refer to $\tau(\bar{x},\bar{y})$ as simply τ.

Given $\mathscr{E}(k) = (\langle t_k \rangle, \mathscr{C}_k, n_{k+1})$.

If $\mathscr{L}(n_k) \neq \underline{\text{HALT}}$ we define

$$\mathscr{E}(k+1) = (\langle t_{k+1} \rangle, \mathscr{C}_{k+1}, n_{k+1}),$$

where for arbitrary y_p and y_q in $\bar{y}$ either

---

4. $\langle d \rangle$ can be viewed as simply an element $(a_{\sigma_1},\ldots,a_{\sigma_n})$ of Σ from Section 2.

(i) $\mathcal{L}(n_k) \in \mathfrak{A}$, say $y_p \leftarrow \tau$, then

$$n_{k+1} = \Delta(n_k),$$

$$\mathscr{C}_{k+1} = \mathscr{C}_k,$$

and

$$t_{k+1} = \begin{cases} t_k, & \text{when } p \neq q, \\ \tau\{t_k/y_\alpha\}^5, & \text{when } p = q, \end{cases}$$

or

(ii) $\mathcal{L}(n_k) \in \mathfrak{B}$, say $\beta_\lambda(\bar{x},\bar{y})$, then

for $\Delta(n_k) = \langle y,z \rangle$ we have

$$n_{k+1} = \begin{cases} y, & \text{when } \bar{\xi} \text{ satisfies } \beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}^6, \\ z & \text{otherwise,} \end{cases}$$

$$\mathscr{C}_{k+1} = \begin{cases} \mathscr{C}_k \cup \{\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}\}, & \text{when } n_{k+1} = y, \\ \mathscr{C}_k \cup \{\sim\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}\} & \text{otherwise,} \end{cases}$$

and

$$t_{k+1} = t_k.$$

The function $\mathscr{C}$ defines an <u>execution sequence</u>

$$\langle \underline{P}, \mathscr{I}, \varphi, \bar{\xi} \rangle$$

---

5. $\tau\{t_k/y_\alpha\}$ $(\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\})$ means each y_α in τ $(\beta_\lambda(\bar{x},\bar{y}))$ is to be replaced by the corresponding t_α of t_k, i.e., $\{t_1/y_1, \ldots, t_n/y_n\}$.

6. It is true that $\bar{\xi}$ either satisfies or does not satisfy $\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}$.

for

$$(\underline{P},\mathcal{I},\varphi,\overline{\xi})$$

with the value of the computation, denoted

$$\mathrm{Val}(\langle \underline{P},\mathcal{I},\varphi,\overline{\xi}\rangle),$$

equal to

$$\rho_{\underline{\xi}}(t_{\Omega}),$$

where

$$\Omega = \max(\mathrm{domain}(\mathcal{C})),$$

when $\mathcal{C}$ is finite (i.e. the program terminates); otherwise

$$\mathrm{Val}(\langle \underline{P},\mathcal{I},\varphi,\overline{\xi}\rangle)$$

is undefined.

We remark that $\rho_{\underline{\xi}}$ applied to t_{Ω} yields a specific vector of constants, i.e.

$$\rho_{\underline{\xi}}(t_{\Omega}) = \langle d\rangle.$$

4. Case Descriptions and Related Concepts.

In the next two sections we extend and formalize some ideas discussed by Pratt in [19,20] in the direction of developing a formal yet practical basis for generating assertions for programs.

4.1 Descriptors. For the abstract program $(\underline{AP},\varphi)$ let δ_i represent a possible sequence of nodes, i.e. a path, $n_{i_1}, n_{i_2}, \ldots, n_{i_n}$ from $(\underline{AP},\varphi)$ having the following properties:

1. $\delta_i(1) = \Delta(\underline{START})$.
2. Given $\delta_i(k)$:
   (a) If $\mathcal{L}(\delta_i(k)) \in \mathcal{A}$ then $\delta_i(k+1) = \Delta(\delta_i(k))$.
   (b) If $\mathcal{L}(\delta_i(k)) \in \mathcal{B}$ and $\Delta(\delta_i(k)) = \langle y,z \rangle$ then
   $\delta_i(k+1) = y$ or $\delta_i(k+1) = z$.
3. There is a maximum η s.t. $\delta_i(\eta) = \underline{HALT}$.

This defines a path $\delta_i \in \{\delta_1, \delta_2, .., \delta_n, \ldots\}$, the set of all possible sequences of nodes (paths) through $(\underline{AP},\varphi)$.

For each δ_i we record in sequence $\varphi(\bar{x})$ followed by

$$\mathcal{L}(n_{i_j}), \qquad j = 1,\ldots,\eta.$$

If δ_i contains an

$$\mathcal{L}(n_{i_j}) \in \mathcal{B}$$

with $\Delta(n_{i_j}) = \langle y,z \rangle$, we record

$$\mathcal{L}(n_{i_j}) \quad \text{or} \quad \sim\mathcal{L}(n_{i_j})$$

depending on whether or not

$$n_{i_{j+1}} = y \quad \text{or} \quad n_{i_{j+1}} = z,$$

respectively. We call this an abstract or unreduced descriptor for $(\underline{AP},\varphi)$.

Definition 2. We define $\mathfrak{D}(\delta_i)$ inductively as follows:

$$\mathfrak{D}(\delta_i)(1) = (\langle t_1 \rangle, \mathscr{C}_1^i, \delta_i(1)),$$

where

$$\mathscr{C}_1^i = \{\varphi(\bar{x})\},$$

and

$$t_1 = x_\nu, \qquad \nu \text{ fixed},$$

if there are input variables, or

$$t_1 = c_\nu, \qquad \nu \text{ fixed},$$

if there are no input variables.

Given

$$\mathfrak{D}(\delta_i)(k) = (\langle t_k \rangle, \mathscr{C}_k^i, \delta_i(k)).$$

For $\mathscr{L}(\delta_i(k)) \neq \underline{\text{HALT}}$ we have

$$\mathfrak{D}(\delta_i)(k+1) = (\langle t_{k+1} \rangle, \mathscr{C}_{k+1}^i, \delta_i(k+1)),$$

where for arbitrary y_p and y_q in $\bar{y}$ either

(i) $\mathscr{L}(\delta_i(k)) \equiv y_p \leftarrow \tau$. In this case we have

$$\mathscr{C}_{k+1}^i = \mathscr{C}_k^i,$$

and

$$t_{k+1} = \begin{cases} t_k, & \text{when } p \neq q, \\ \tau\{t_k/y_\alpha\}, & \text{when } p = q, \end{cases}$$

or

(ii) If $\mathscr{L}(\delta_i(k)) \equiv \beta_\lambda(\bar{x},\bar{y})$, then

for $\Delta(\delta_i(k)) = \langle y,z \rangle$ we have

$$\mathscr{C}^{\,i}_{k+1} = \begin{cases} \mathscr{C}^{\,i}_{k} \cup \{\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}\}, & \text{when } \delta_i(k+1) = y, \\ \text{else } \mathscr{C}^{i}_{k} \cup \{\sim\beta_\lambda(\bar{x},\bar{y})\{t_k/y_\alpha\}\}, \end{cases}$$

and

$$t_{k+1} = t_k.$$

For each δ_i, $\mathfrak{D}(\delta_i)$ produces what is called a reduced descriptor or simply descriptor, denoted as

$$(\underline{D},\varphi)_{\delta_i},$$

with control set

$$\mathscr{C}^{i}_{\omega}$$

and kernel set $\mathscr{K}$ which is

$$\langle t_\omega \rangle,$$

where

$$\omega = \max(\text{domain } \mathfrak{D}(\delta_i)).$$

The main differences between the original path δ_i and its corresponding descriptor is that the descriptor contains no local variables or extraneous sequencing information. A descriptor specifies the sequence of operations only where necessary, i.e. in function composition and variables in argument expressions. This means that we now have a representation for a program path which will in many cases exhibit quite clearly the information necessary to understand what a program (or part of a program) computes if that particular path is followed.

4.2 Case Descriptions. We define an abstract case description, denoted $(\underline{AC},\varphi)$, to be $\bigcup_{i\geq 1} (\underline{D},\varphi)_{\delta_i}$, i.e. we generate a descriptor for each (finite) path δ_i in $(\underline{AP},\varphi)$. Of course (AC,φ) is countably infinite unless $(\underline{AP},\varphi)$ contains no loops.

$(\underline{AC},\varphi)$ together with an interpretation $\mathcal{I}$ forms a case description $(\underline{C},\mathcal{I},\varphi)$.

An interpreted case description is a case description with an assignment $\bar{\xi} \in \underline{D}_{\bar{x}}$ and is written as $(\underline{C},\mathcal{I},\varphi,\bar{\xi})$.

The control sets of the descriptors in the interpreted case description are composed of ground instances of their predicates which can now be evaluated in the usual manner. To find the unique descriptor corresponding to a particular execution sequence we simply search the list of descriptors for the descriptor all of whose control set predicates are satisfied. This descriptor's kernel set specifies the values computed for that execution sequence. However, from a well-known undecidability result we know that it is impossible in general to determine whether an arbitrary program terminates for all inputs. Therefore, we may never find a descriptor in the interpreted case description having all predicates in its control set satisfied, thus continuing to test indefinitely. We formalize this intuitive notion of functional equivalence in the Section 5.

4.3 Examples of Case Descriptions. We now add to our program an output predicate (or output assertion), denoted $\psi(\bar{x},\bar{y})$ which is a wff with no free individual variables other than $\bar{x}$ and $\bar{y}$. $\psi(\bar{x},\bar{y})$ usually specifies the desired relation between the input and output variables. When convenient in an example we add a set of individual output variables $\bar{z} = \{z_1, \ldots, z_n\}$ to distinguish for the reader exactly which variables are being assigned values

by the program. This slight addition makes the output assertion $\psi(\bar{x},\bar{z})$.[7] We henceforth refer to the output assertion as simply ψ as its arguments will be obvious from context.

The following examples illustrate the derivation of case descriptions and their simplistic representation of a program. In addition we show a possible use of case descriptions for either forming a particular ψ or "checking" one that was given a priori. If an inaccurate ψ is provided serious difficulties can arise in an attempt to prove partial correctness (for discussion see [8]). In addition if an inaccurate ψ is used in the partial or total derivation of an inductive assertion, as in Wegbreit [22] and Katz and Manna [12], it is highly unlikely that the loop invariants will ever be found.

---

7. The theorems presented in this work still hold if this addition is made throughout.

In Figure 1 we see a simple factorial program with range 0 to 50. Table 1 contains descriptors of its case description for some of the shorter paths. Each descriptor is shown in its unreduced, reduced, and interpreted (and simplified) form. Note that the number of unreduced and reduced descriptors is countably infinite due to the loop in the abstract program. However, by viewing the interpreted schemata we see that only a finite number of descriptors are possible. Paths which cannot be followed irrespective of the initial assignment yield a contradiction in the control set (denoted by $\square$) as shown by interpreted descriptor 52. The importance of this will become magnified in more complex programs.

In obtaining the interpreted descriptor we can employ a limited theory of types and specialized routines for algebraic simplification and solving linear inequalities. Since in the control set, with possibly the exception of $\varphi(\bar{x})$, we are dealing only with skolem constants we expect very efficient analysis. Some of the specific techniques we will use are discussed by Bledsoe et al. in [2].

Looking again at Table 1, by viewing only a few descriptors we can easily see that ψ is $\{Z=N!\}$.

Notation. Let $\delta(i_1, \ldots, i_n)$ denote a path (or subpath) through the flowchart over arcs $i_1, \ldots, i_n$. We henceforth omit from our tables the unreduced descriptor adding instead the statement $\delta_j = \delta(i_1,\ldots,i_n)$ to the descriptor.

Figure 2 (taken from [8]) computes the fractional quotient P/Q to within tolerance E.[8] In Table 2 we see the (partial) case description for Figure 2. It's easy to see from the interpreted descriptors that ψ is $\{P/Q - E < Z \leq P/Q\}$.

---

8. In this example the domain is the reals.

In Figure 3 (taken from [13]) we have a program which multiplies two numbers, accepts signed inputs, and all additive operations are restricted to incrementing and decrementing by one. From Table 3 it is easily seen that ψ is $\{Y = DA * B\}$. As in Table 1 we again uncovered a path which cannot be followed, viz. δ_4. However, we note that in this instance it was not entirely obvious that δ_4 could not be executed. The knowledge that a particular path cannot be followed, even if $\varphi(\bar{x})$ is satisfied, will in many cases prove to be quite valuable information to other components of the verification system.

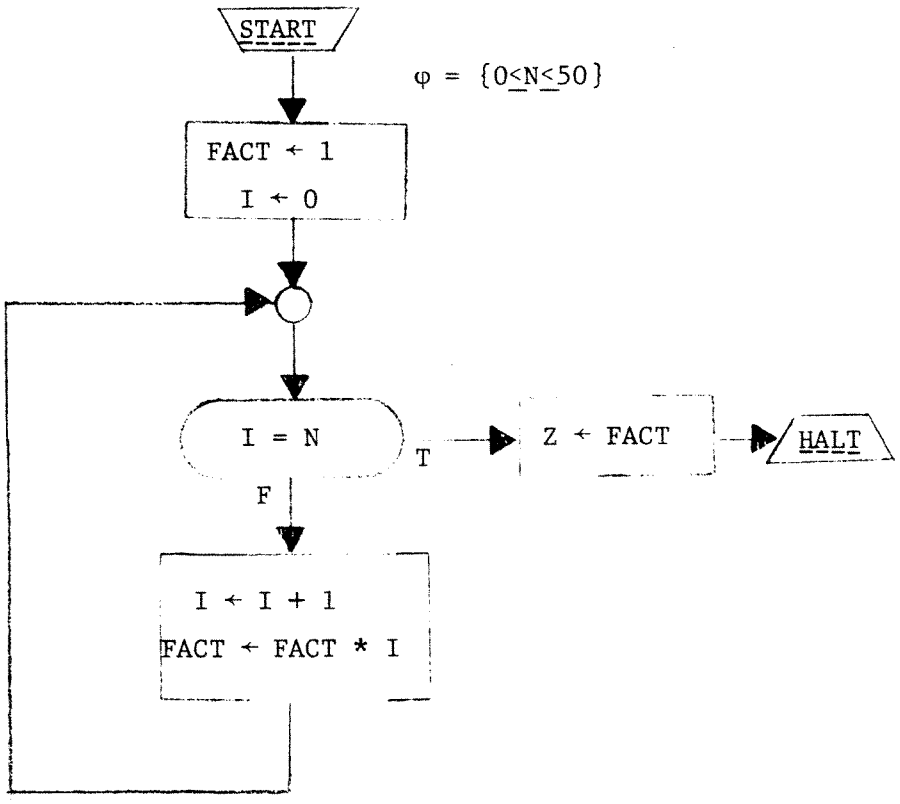


FIGURE 1. FACTORIAL PROGRAM.

	Unreduced Descriptor	Descriptor	Interpreted Descriptor
1	(0≤N≤50); FACT←1; I←0; (I=N); Z←FACT	$\mathscr{C}$ = {0≤N≤50; 0=N} $\mathscr{K}$ = {Z←1}	$\mathscr{C}$ = {N=0} $\mathscr{K}$ = {Z←1}
2	(0≤N≤50); FACT←1; I←0; ∿(I=N); I←I+1; FACT←FACT*I; (I = N); Z←FACT	$\mathscr{C}$ = {0≤N≤50, ∿(0=N), 0+1=N} $\mathscr{K}$ = {Z←1*(0+1)}	$\mathscr{C}$ = {N=1} $\mathscr{K}$ = {Z←1}
3	(0≤N≤50); FACT←1; I←0; ∿(I=N); I←I+1; FACT←FACT*I; ∿(I=N); I←I+1; FACT←FACT*I; (I=N); Z←FACT	$\mathscr{C}$ = {0≤N≤50, ∿(0=N), ∿(0+1=N), (0+1) + 1=N} $\mathscr{K}$ = Z ← (1*(0+1))* ((0+1) + 1)}	$\mathscr{C}$ = {N=2} $\mathscr{K}$ = {Z←2}
	. . .	. . .	. . .
52	(0≤N≤50): FACT←1; I←0; [∼(I=N); I←I+1; FACT←FACT*I]^{51}; (I=N); Z←FACT	$\mathscr{C}$ = {0≤N≤50, ∿(N=0), ..., (N=50), N=51} $\mathscr{K}$ = ∼Z←1*1*2*...*51}	□

Table 1: Case Description of Factorial Program in Figure 1.

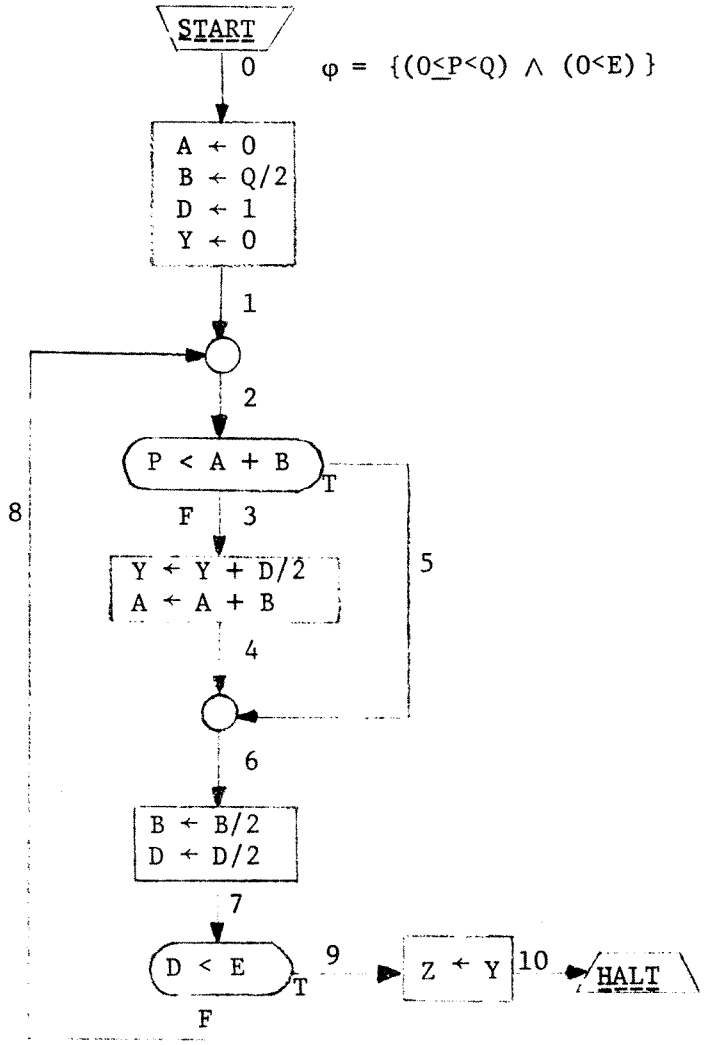


Figure 2. Wensley's Quotient Algorithm.

Descriptor	Interpreted Descriptor
$\delta_1 = \delta(0,1,2,5,6,7,9,10)$ $\mathscr{C} = \{0 \leq P < Q,\ 0 < E,\ P < 0+Q/2,$ $1/2 < E\}$ $\mathscr{K} = \{Z \leftarrow 0\}$	$\mathscr{C} = \{P/Q - E < 0 \leq P/Q < 1/2\}$ $\mathscr{K} = \{Z \leftarrow 0\}$
$\delta_2 = (0,\ldots,7,9,10)$ $\mathscr{C} = \{0 \leq P < Q,\ 0 < E,\ \sim(P < 0+Q/2),$ $1/2 < E\}$ $\mathscr{K} = \{Z \leftarrow (0+1/2)\}$	$\mathscr{C} = \{P/Q - E < 1/2 \leq P/Q < 1$ $\mathscr{K} = \{Z \leftarrow 1/2\}$
$\delta_3 = (0,1,2,5,6,7,8,2,5,6,7,9,10)$ $\mathscr{C} = \{0 \leq P < Q,\ 0 < E,\ P < 0+Q/2,$ $\sim(1/2 < E),\ P < 0+(Q/2)/2,$ $(1/2)/2 < E\}$ $\mathscr{K} = \{Z \leftarrow 0\}$	$\mathscr{C} = \{P/Q - E < 0 \leq P/Q < 1/4$ $\mathscr{K} = \{Z \leftarrow 0\}$
. . .	. . .

Table 2: (Partial) Case Description for Wensley Quotient Algorithm.

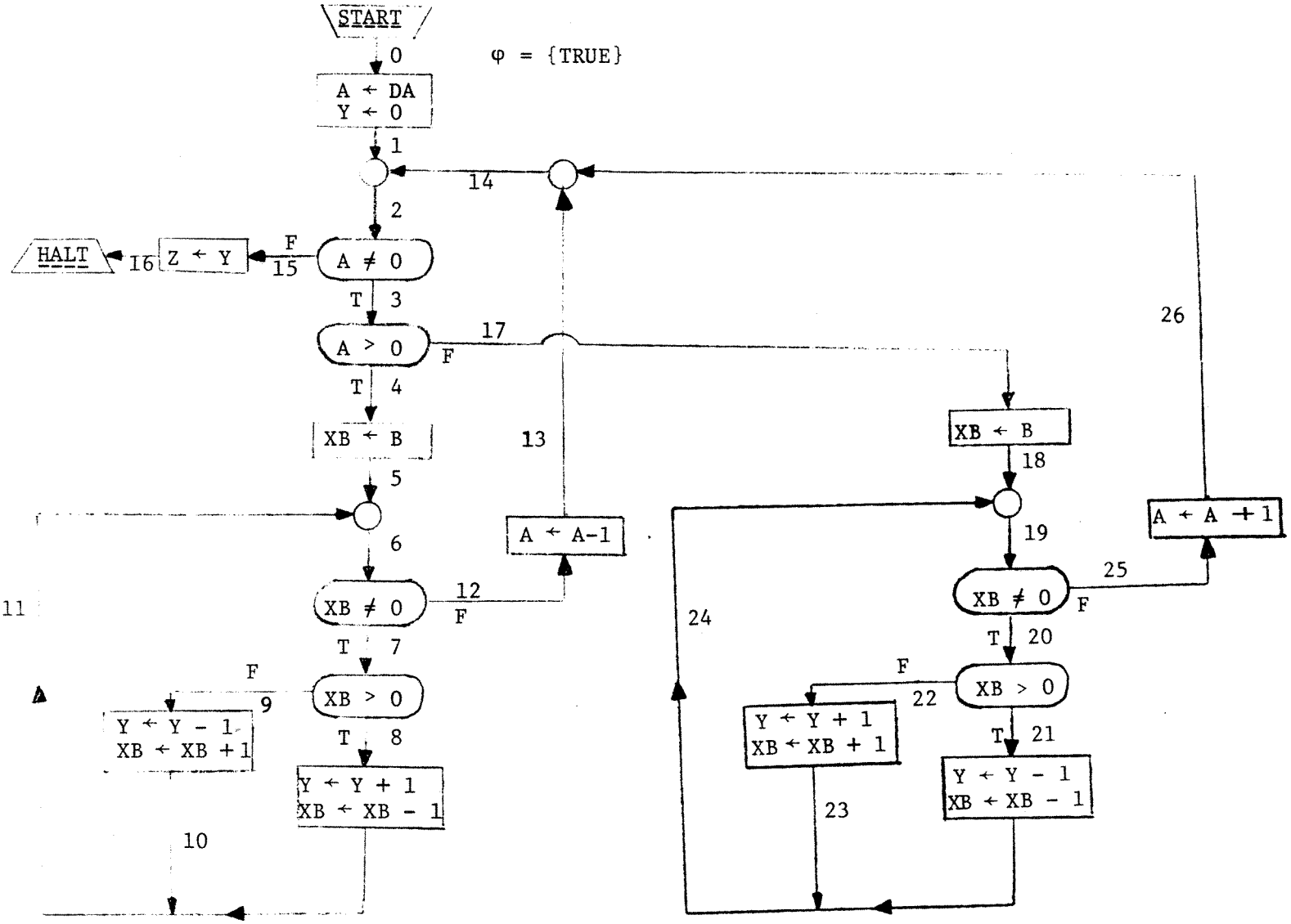


FIGURE 3. MULTIPLICATION PROGRAM.

Descriptor	Interpreted Descriptor
$\delta_1 = \delta(0,\ldots,6,12,13,14,2,15,16)$ $\mathscr{C} = \{DA\neq 0,\ DA>0,\ \sim(B\neq 0),\ \sim(DA-1\neq 0)\}$ $\mathscr{K} = \{Z\leftarrow 0\}$	$\mathscr{C} = \{B=0,\ DA=1\}$ $\mathscr{K} = \{Z\leftarrow 0\}$
$\delta_2 = \delta(0,\ldots,8,11,6,12,13,14,2,15,16)$ $\mathscr{C} = \{DA\neq 0,\ DA>0,\ B\neq 0,\ B>0,\ \sim(B-1\neq 0),$ $\sim(DA-1\neq 0)\}$ $\mathscr{K} = \{Z\leftarrow(0+1)\}$	$\mathscr{C} = \{B=1,\ DA=1\}$ $\mathscr{K} = \{Z\leftarrow 1\}$
$\delta_3 = \delta(0,\ldots,3,17,\ldots,21,24,19,25,26,$ $14,2,15,16)$ $\mathscr{C} = \{DA\neq 0,\ \sim(DA>0),\ B\neq 0,\ B>0,\ \sim(B-1\neq 0)$ $\sim(DA+1\neq 0)\}$ $\mathscr{K} = \{Z\leftarrow(0-1)\}$	$\mathscr{C} = \{B=1,\ DA=-1\}$ $\mathscr{K} = \{Z\leftarrow -1\}$
$\delta_4 = \delta(0,\ldots,8,11,6,7,9,10,11,6,12,13,$ $14,2,15,16)$ $\mathscr{C} = \{DA\neq 0,\ DA>0,\ B\neq 0,\ B>0,\ B-1\neq 0,$ $\sim(B-1>0),\ \sim((B-1)+1\neq 0),$ $\sim(DA-1\neq 0)\}$ $\mathscr{K} = \{Z\leftarrow((0+1)-1)\}$	$\square$
$\delta_5 = \delta(0,\ldots,8,11,6,7,8,11,6,12,13,14,2,$ $15,16)$ $\mathscr{C} = \{DA\neq 0,\ DA>0,\ B\neq 0,\ B>0,\ B-1\neq 0$ $\sim((B-1)-1\neq 0),\ \sim(DA-1\neq 0)\}$ $\mathscr{K} = \{Z\leftarrow(0+1)+1\}$	$\mathscr{C} = \{B=2,\ DA=1\}$ $\mathscr{K} = \{Z\leftarrow 2\}$
. . .	. . .

Table 3. (Partial) Case Description for Multiplication Program in Figure 3.

Definition 4. $(\underline{AC},\varphi)$ is said to be consistent if and only if for every $\bar{\xi} \in \underline{D}_{\underline{x}}$ there is at most one descriptor

$$(\underline{D},\varphi)_{\delta_i}$$

in

$$(\underline{C},\mathcal{I},\varphi,\bar{\xi})$$

such that all of its control set predicates are satisfied by $\bar{\xi}$.

Definition 5. If $(\underline{AC},\varphi)$ is consistent then the value of $(\underline{C},\mathcal{I},\varphi,\bar{\xi})$, written

$$\text{Val}((\underline{C},\mathcal{I},\varphi,\bar{\xi})),$$

is determined as follows:

1. Let $(\underline{D},\varphi)_{\delta_i}$ be the unique descriptor in $(\underline{AC},\varphi)$ having all of its control set predicates satisfied by $\mathcal{I}$ and $\bar{\xi}$. In this case

$$\text{Val}((\underline{C},\mathcal{I},\varphi,\bar{\xi})) = \mathcal{K}((\underline{D},\varphi)_{\delta_i}),$$

the kernel set of $(\underline{D},\varphi)_{\delta_i}$ under $\mathcal{I}$ and $\bar{\xi}$.

2. If no such $(\underline{D},\varphi)_{\delta_i}$ exists then

$$\text{Val}((\underline{C},\mathcal{I},\varphi,\bar{\xi}))$$

is undefined.

We are now ready to prove

Lemma 1. (Consistency Lemma) If $(\underline{AC},\varphi)$ is an abstract case description then $(\underline{AC},\varphi)$ is consistent.

Proof. Suppose $\mathcal{I}$ and $\bar{\xi}$ satisfy $(\underline{AC},\varphi)$ such that there are two distinct descriptors

$$(\underline{D},\varphi)_{\delta_i}$$

5. Functional Equivalence of Interpreted Programs and Case Descriptions.

We begin by making the following definitions.

Definition 3. $\mathscr{I}$ and $\bar{\xi} \in \underline{D}_{\bar{X}}$ satisfies $(\underline{AC},\varphi)$ means that there is a descriptor

$$(\underline{D},\varphi)_{\delta_i} \in (\underline{AC},\varphi)$$

such that $\mathscr{I}$ and $\bar{\xi}$ satisfy

$$\mathscr{C}^{i}_{\omega}$$

where

$$\omega = \max \{\text{domain } \mathfrak{D}(\delta_i)\}.$$

and

$$(\underline{D},\varphi)_{\delta_j}$$

with $\mathscr{C}^i_\omega$ and $\mathscr{C}^j_\omega$, respectively, satisfied by $\mathscr{S}$ and $\bar{\xi}$. Let

$$\mu = \min\{\gamma \mid \delta_i(\gamma) \neq \delta_j(\gamma)\}.$$

Therefore

$$\delta_i(\mu-1) = \delta_j(\mu-1).$$

Now consider the case when

$$\delta_i(\mu-1)$$

is an assignment node. From the definition of path we have

$$\delta_i(\mu) = \Delta(\delta_i(\mu-1)) = \Delta(\delta_j(\mu-1)) = \delta_j(\mu)$$

which cannot be the case. Thus $\delta_i(\mu-1)$ must be a branch node.

Since $\delta_i(\mu-1)$ is a branch node and

$$\delta_i(\mu) \neq \delta_j(\mu),$$

without loss of generality we may assume that

$$\delta_i(\mu) = y$$

and

$$\delta_j(\mu) = z$$

for $\Delta(\delta_i(\mu-1)) = \langle y,z \rangle$. Thus

$$\mathscr{C}^i_\mu \supseteq \mathscr{L}(\delta_i(\mu-1))\{t_{\mu-1}/y_\alpha\} \tag{1}$$

and

$$\mathscr{C}^j_\mu \supseteq \sim\mathscr{L}(\delta_i(\mu-1))\{t_{\mu-1}/y_\alpha\}. \tag{2}$$

But now

$$\mathscr{C}^i_\mu \subseteq \mathscr{C}^i_{\max(\text{domain } \mathfrak{D}(\delta_i))}$$

and

$$\mathscr{C}^{j}_{\mu} \subseteq \mathscr{C}^{j}_{\max(\text{domain } \mathcal{D}(\delta_j))},$$

and since

$$\mathscr{C}^{i}_{\max(\text{domain } \mathcal{D}(\delta_i))}$$

and

$$\mathscr{C}^{j}_{\max(\text{domain } \mathcal{D}(\delta_j))}$$

are satisfied by $\mathcal{I}$ and $\bar{\xi}$ by assumption, it follows that $\mathcal{I}$ and $\bar{\xi}$ satisfy $\mathscr{C}^{i}_{\mu}$ and $\mathscr{C}^{j}_{\mu}$ which contradicts (1) and (2). Q.E.D.

Lemma 2. $\mathcal{I}$ and $\bar{\xi}$ satisfy

$$(\underline{AC},\varphi)$$

if and only if

$$\langle \underline{P},\mathcal{I},\varphi,\bar{\xi}\rangle$$

is defined.

Proof. If $\mathcal{I}$ and $\bar{\xi}$ satisfy

$$(\underline{AC},\varphi)$$

it follows from Lemma 1 that there is a unique descriptor

$$(\underline{D},\varphi)_{\delta_i} \in (\underline{AC},\varphi)$$

such that its control set $\mathscr{C}^{i}_{\omega}$ is satisfied. We recall that

$$\varphi(\bar{x}) \in \mathscr{C}^{i}_{\omega}$$

and is thus satisfied by assumption. It can be shown by induction that $\mathscr{C}_k = \mathscr{C}^{i}_{k}$ for $i = 1,\ldots,\omega$. We also have $\omega = \Omega$. Thus

$$\langle \underline{P},\mathcal{I},\varphi,\bar{\xi}\rangle$$

must follow δ_i which means that it is defined.

Suppose

$$\langle \underline{P}, \mathcal{I}, \varphi, \overline{\xi} \rangle$$

is defined. Since there is some finite path δ_i followed by

$$\langle \underline{P}, \mathcal{I}, \varphi, \overline{\xi} \rangle,$$

it follows directly that there is a

$$(\underline{D}, \varphi)_{\delta_i} \in (\underline{\underline{AC}}, \varphi)$$

with its control set satisfied. Q.E.D.

We now have

Corollary 1. $\mathcal{I}$ and $\overline{\xi}$ do not satisfy

$$(\underline{\underline{AC}}, \varphi)$$

if and only if

$$\langle \underline{P}, \mathcal{I}, \varphi, \overline{\xi} \rangle$$

is undefined.

We are now in a position to prove the main results.

Theorem 1. If $\mathcal{I}$ and $\overline{\xi}$ satisfy

$$(\underline{\underline{AC}}, \varphi)$$

then

$$\mathrm{Val}((\underline{C}, \mathcal{I}, \varphi, \overline{\xi})) = \mathrm{Val}(\langle \underline{P}, \mathcal{I}, \varphi, \overline{\xi} \rangle).$$

Proof. By hypothesis and the Consistency Lemma we know that there is a unique descriptor

$$(\underline{D}, \varphi)_{\delta_i} \in (\underline{\underline{AC}}, \varphi)$$

such that its control set $\mathscr{C}_\omega^i$ is satisfied by $\mathcal{I}$ and $\overline{\xi}$. From Lemma 2

$$\langle \underline{P}, \mathcal{I}, \varphi, \overline{\xi} \rangle$$

is defined and furthermore must follow δ_i. Since it can easily be shown

that

$$\langle t_\omega \rangle = \langle t_\Omega \rangle ,$$

we must have

$$\rho_{\underline{\xi}}(t_\omega) = \rho_{\underline{\xi}}(t_\Omega).$$

Therefore

$$\mathrm{Val}((\underline{C},\mathcal{I},\varphi,\overline{\xi})) = \mathrm{Val}\ (\langle \underline{P},\mathcal{I},\varphi,\overline{\xi}\rangle).\quad \underline{Q}.\underline{E}.\underline{D}.$$

By a similar argument we get

Theorem 2. If

$$\langle \underline{P},\mathcal{I},\varphi,\overline{\xi}\rangle$$

is defined then

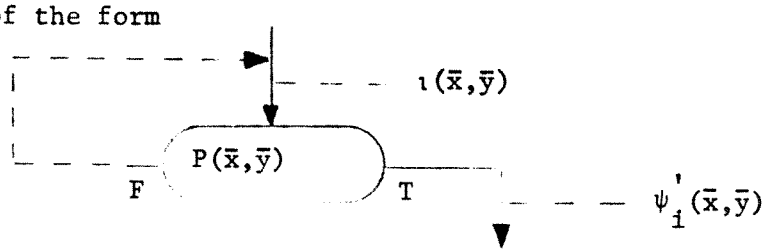
$$\mathrm{Val}((\underline{C},\mathcal{I},\varphi,\overline{\xi})) = \mathrm{Val}(\langle \underline{P},\mathcal{I},\varphi,\overline{\xi}\rangle).$$

As a result of Corollary 1, Theorem 1, and Theorem 2 we have now established the functional equivalence of an interpreted program and its corresponding interpreted case description.

6. Heuristics for Generating Inductive Assertions.

We now turn our attention to the development of heuristics which in some cases suffice by themselves to generate inductive assertions; but more often, for non-trivial programs, are used in concert with a (partial) case description. Many of our heuristic techniques are similar in content to those of Wegbreit [22] and Katz and Manna [12]. A significant difference, however, is that we develop only a few heuristic rules, of which normally only one applies, such that when they are employed (possibly in conjunction with a (partial) case description) a consistent assertion is usually formed on the first try. Hence, we generally avoid fruitless attempts at generating and trying to prove verification conditions for inconsistent predicates.

For simplicity in the statement of the heuristic rules, we view an exit from a loop to be of the form



where the cutpoint for the loop immediately preceeds the exit test and is marked by $\iota(\bar{x},\bar{y})$ representing the inductive assertion for the loop, $P(\bar{x},\bar{y})$ is the exit test, and $\psi_i'(\bar{x},\bar{y})$ is some conjunct of a predicate $\psi'(\bar{x},\bar{y})$ known to be true when $P(\bar{x},\bar{y})$ is satisfied. We observe that $\psi'(\bar{x},\bar{y})$ is usually different from $\psi(\bar{x},\bar{y})$. However, in the simplest case $\psi'(\bar{x},\bar{y})$ is merely $\psi(\bar{x},\bar{y})$, i.e. there are no statements between $\psi(\bar{x},\bar{y})$ and the branch statement under consideration.

We now consider the following heuristics which are applied in the specified order to the formulas in their "natural" form.

1. Convert $\psi'(\bar{x},\bar{y})$ to conjunctive form, i.e. $\psi'(\bar{x},\bar{y}) = \psi_1'(x,y) \wedge \ldots \wedge \psi_n'(\bar{x},\bar{y})$, and consider each $\psi_i'(\bar{x},\bar{y})$ separately with the goal being to form

for each $\psi_i'(\bar{x},\bar{y})$ an appropriate $\iota_k(\bar{x},\bar{y})$ where $\iota(\bar{x},\bar{y}) = \iota_1(\bar{x},\bar{y}) \wedge \ldots \wedge \iota_\ell(\bar{x},\bar{y})$.

2. (Transitivity) If $\psi_i'(\bar{x},\bar{y})$ is of the form $t_1 R_1 t_2$ and $P(\bar{x},\bar{y})$ is of the form $t_3 R_2 t_4$, where t_1,t_2,t_3,t_4 are terms and R_1,R_2 are inequality relations, then find the appropriate $\iota_k(\bar{x},\bar{y})$ s.t. $\iota_k(\bar{x},\bar{y}) \Rightarrow (P(\bar{x},\bar{y}) \Rightarrow \psi_i'(\bar{x},\bar{y}))$. To do this we normally employ the usual transitive closure properties with the domain being the integers unless otherwise specified. If $\psi_i'(\bar{x},\bar{y})$ contains quantification we apply a transitivity axiom directly to $P(\bar{x},\bar{y}) \Rightarrow \psi_i'(\bar{x},\bar{y})$ and if necessary make the appropriate subscript changes when arrays occur within the quantified expression. (See the second example in Section 7.) If no transitive rule applies to R_1 and R_2 we simply choose $\iota_k(\bar{x},\bar{y}) = \psi_i'(\bar{x},\bar{y})$.

3. (Equality) This rule is generally used only in conjunction with a case description. The reason for this is to avoid making an erroneous assertion which in this case happens quite easily. We make this description definitive by viewing a few specific cases.

(a) If $P(\bar{x},\bar{y})$ is an equality branch with $P(\bar{x},\bar{y}) \neq \psi_i'(\bar{x},\bar{y})$ and $P(\bar{x},\bar{y})$ is of the form $t_i = 0$, then temporarily let $\iota_k(\bar{x},\bar{y}) = \psi_i'(\bar{x},\bar{y})$ and compute a (partial) case description for the loop. We use this (partial) case description to determine where the t_i occurs in $\iota_k(\bar{x},\bar{y})$. For example, suppose we have $A = 0 \Rightarrow X = Y * Z$. There are numerous plausible $\iota_k(\bar{x},\bar{y})$ such as $X + A = Y * Z$, $X * C \uparrow A = Y * Z$, $X * (C * A) = Y * Z$, $X * C \uparrow A = Y * Z * C \uparrow A$ and so on where C is a constant. We avoid this proliferation of possible inductive assertions by generating a (partial) case description which in many cases similar to this produce a consistent inductive assertion without human intervention. (See the third example in Section 7.)

(b) If $P(\bar{x},\bar{y})$ is an equality branch with $P(\bar{x},\bar{y}) \neq \psi_i'(\bar{x},\bar{y})$ and is of the form $t_i = t_j$ where $t_i \neq t_j \neq 0$, we first try $\iota_k(\bar{x},\bar{y}) = \psi_i''(\bar{x},\bar{y})$

where $\psi_i''(\bar{x},\bar{y})$ is obtained from $\psi'(\bar{x},\bar{y})$ by replacing all occurences of t_i by t_j, and then if that fails, try replacing t_j by t_i. This appears to be sufficient for many applications, but if it is not we again revert to a (partial) case description to guide further substitutions.

4. If $P(\bar{x},\bar{y}) \neq \psi_i'(\bar{x},\bar{y})$, let $\iota_k(\bar{x},\bar{y}) = \psi_i'(\bar{x},\bar{y})$.

5. Let $\iota_k(\bar{x},\bar{y})$ be $P(\bar{x},\bar{y}) \Rightarrow \psi_i'(\bar{x},\bar{y})$.

Rules 4 and 5 will normally be considered as yielding trial inductive assertions for which we will generate a (partial) case description to verify the correctness of $\iota_k(\bar{x},\bar{y})$ before generating any verification conditions.

We reiterate at this point that these rules simply exhibit the general flavor of the approach rather than a detailed analysis of all heuristic rules which are employed.

7. Generation of Inductive Assertions via Case Descriptions and Heuristic Rules.

We simply modify our method for generating a (partial) case description so that we may now direct our attention only at specific parts of a program, viz. loops in the program requiring inductive assertions. We now want a descriptor to reflect only what is computed for a particular path in the loop under consideration and not for a path through the entire program as in Section 4. To do this we locate the "nearest" assertion which is on a path into the loop and begin generating descriptors in the usual fashion beginning at this point. Essentially we are simply viewing this presumably valid wff (with $\underline{D}$ being either specifically or vacuously supplied by $\varphi(\bar{x})$) as a new φ. When appropriate equality predicates in this new φ which contain individual program variables can be deleted from φ and viewed as assignment statements. We now view as individual output variables the local variables of the loop which occur on the left side of a statement of the form $y_i \leftarrow \tau$. Rather than terminating the case description at the HALT node, we terminate immediately after exiting the loop.

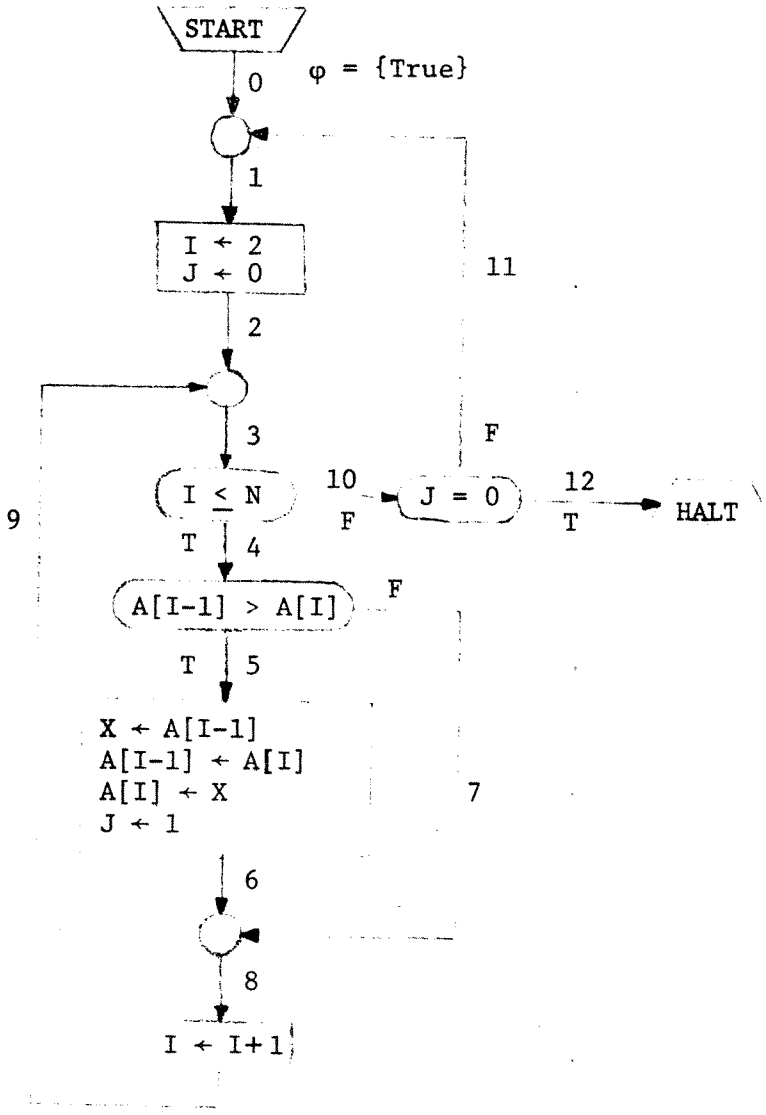
Notation. We assume that by now the reader is familiar with the derivation of a case description. As a result we henceforth present only the interpreted (and simplified) descriptor in the tables.

Looking at Figure 4 (taken from [13]) we see a simple exchange sort program with $\psi = \forall M(2 \leq M \leq N \Rightarrow A(M-1) \leq A(M))$. By taking ψ backwards over path $\delta(3,10,12)$ we get $I > N \rightarrow (J=0 \rightarrow \forall M(2 \leq M \leq N \rightarrow A(M-1) \leq A(M)))$. Applying the rewrite rules of IMPLY [3], which is a natural-deduction-type system which processes formulas in their "natural" form, immediately yields $I > N \wedge J = 0 \Rightarrow \forall M(2 \leq M \leq N \Rightarrow A(M-1) \leq A(M))$. Using Rule 2 we get $J = 0 \Rightarrow \forall M(2 \leq M \leq I \rightarrow A(M-1) \leq A(M))$. In practice we would probably verify this choice of ι by

producing a (partial) case description which computes exactly this trial ι. This choice turns out to be a consistent inductive assertion for the program in Figure 4.

For the Wensley Division Algorithm of Figure 2 we try to find an appropriate ι at arc 5. We begin by applying Rule 1 to the ψ we found in Section 3 thus obtaining $\psi_1 = P/Q - E < Z$ and $\psi_2 = Z \leq P/Q$. Dragging ψ_1 backwards to arc 5 we get $D < E \Rightarrow P/Q - E < Y$. Rule 2 now gives $\iota_i = P/Q - D \leq Y$. For ψ_2 we get $D < E \Rightarrow Y \leq P/Q$ and since no transitive property applies by Rule 2 we get $\iota_2 = Y \leq P/Q$. We now observe local variables A and B which are not in ι_1 or ι_2. We thus form a (partial) case description for the loop to establish loop invariants containing A and B. From Table 4 we can easily see that $A = Q * Y$ and $B = (Q*D)/2$. We now have the loop invariant $\iota = \iota_1 \wedge \iota_2 \wedge \iota_3 \wedge \iota_4$ which yields valid verification conditions.

In Figure 3 we saw a fairly complex multiplication program. We now find the inductive assertions ι_2, ι_6, and ι_{19} to be positioned at arcs 2,6 and 19. Backing up ψ over $\delta(2,15,16)$ we get $A = 0 \Rightarrow Y = B * DA$. Rule 3 picks $\iota_2 = \{Y=B*DA\}$ and we then generate Table 5. From this Table we have a technique which immediately yields $\iota_2 = \{Y=B*(DA-A)\}$. We now try to find ι_6 by first backing up ι_2 over path $\delta(6,12,13,14)$ giving $XB = 0 \Rightarrow Y = B * (DA-A+1)$. Again Rule 3 picks $\iota_6 = \{Y=B*(DA-A+1)\}$ and generates Table 6 which directly gives $\iota_6 = \{Y=B*(DA-A)+(B-XB)\}$. It turns out that ι_{19} also follows easily yielding $\iota_{19} = \{Y=B*(DA-A)-(B-XB)\}$.



$$\psi \quad = \quad \forall M(2 \leq M \leq N \Rightarrow A[M-1] \leq A[M])$$

FIGURE 4. SIMPLE EXCHANGE SORT.

Interpreted Descriptor
$\delta_1 = \delta(0,1,2,5,6,7,9)$ $\mathscr{C} = \{0 \leq P < Q,\ E > 1/2\}$ $\mathscr{K} = \{A \leftarrow 0,\ B \leftarrow Q/4,\ D \leftarrow 1/2,\ Y \leftarrow 0\}$
$\delta_2 = \delta(0,\ldots,4,6,7,9)$ $\mathscr{C} = \{Q/2 \leq P < Q,\ E > 1/2\}$ $\mathscr{K} = \{A \leftarrow Q/2,\ B \leftarrow Q/4,\ D \leftarrow 1/2,\ Y \leftarrow 1/2\}$
$\delta_3 = \delta(0,1,2,5,6,7,8,2,5,6,7,9)$ $\mathscr{C} = \{0 \leq P < Q/2,\ 1/4 < E \leq 1/2\}$ $\mathscr{K} = \{A \leftarrow 0,\ B \leftarrow Q/8,\ D \leftarrow 1/4,\ Y \leftarrow 0\}$
$\delta_4 = \delta(0,1,2,5,\ldots,8,3,4,6,7,9)$ $\mathscr{C} = \{Q/4 \leq P < Q/2,\ 1/4 < E \leq 1/2\}$ $\mathscr{K} = \{A \leftarrow Q/4,\ B \leftarrow Q/8,\ D \leftarrow 1/4,\ Y \leftarrow 1/4\}$
. . .

Table 4. (Partial) Case Description for Loop in Wensley Algorithm of Figure 2.

Interpreted Descriptor
δ_1 = δ(0,...,8,11,6,12,13,14,2,15) $\mathscr{C}$ = {DA=1, B=1} $\mathscr{K}$ = {Y←1, B←1, A←DA-1}
δ_2 = δ(0,...,8,11,6,7,8,11,6,12,13,14,2,15) $\mathscr{C}$ = {DA=1, B=2} $\mathscr{K}$ = {Y←2, B←2, A←DA-1}
. . .

Table 5.

Interpreted Descriptor
δ_1 = δ(2,...,8,11,6,12) $\mathscr{C}$ = {A≥1, B=-1} $\mathscr{K}$ = {Y←B*(DA-A)+1, XB←B-1}
δ_2 = δ(2,...,7,9,10,11,6,12) $\mathscr{C}$ = {A≥1, B=-1} $\mathscr{K}$ = {Y←B*(DA-A)-1, XB←B+1}
. . .

Table 6.

Tables 5 and 6 are from the Multiplication Program of Figure 3.

8. Conclusion.

The techniques described in this paper are presently being implemented by the author on a CDC 6600/6400 computer system via the U. T. time-sharing system SATURN.

At present we expect to develop an interactive system allowing the human to recognize "patterns" in the descriptors. This human intervention would normally occur only when our heuristic rules don't apply or when the appropriate assertion becomes obvious to the user thus making it expeditious to terminate the generation process. It is clear, however, that more and better heuristic rules are needed. We anticipate new rules which interface smoothly with (partial) case descriptions to surface as we gain experience with the system.

The theorem-proving required is within the scope of some present automatic theorem proving systems which are adept at simplification and proving verification conditions (see e.g. Bledsoe [1] or Deutsch [6]).

We emphasize that we do not foresee the proposed techniques bringing to fruition the practical verification of large, complex programs which now exist and were written in an arbitrary fashion. However, we do feel that the ideas developed here will carry over into an environment in which the programmer writes hierarchically well-structured programs creating the Floyd assertions along with the program rather than ex post facto. We anticipate that in this case our tools will provide a practical basis for the verification of large, complicated programs.

Acknowledgements. I am deeply indebted to my supervisor Professor W. W. Bledsoe and to Drs. Dallas Lankford and Terrence W. Pratt.

REFERENCES

1. Bledsoe, W. W., "Program correctness." University of Texas Departmental Memo, Austin, January, 1974.

2. Bledsoe, W. W.; R. S. Boyer; and W. H. Henneman, "Computer proofs of limit thoerems." Artificial Intelligence 3 (1972), 27-60.

3. Bledsoe, W. W.; and Peter Bruell, "A man-machine theorem proving system." IJCAI-3 (August 1973), 56-65.

4. Church, A., Introduction to mathematical logic. Vol. 1, Princeton University Press, Princeton, N.J., 1956.

5. Cooper, D. C., " Programs for mechanical program verification." Machine Intelligence 6, American Elsevier (1971), 43-59.

6. Deutsch, L. P., "An interactive program verifier." Ph.D. Dissertation, University of California, Berkeley, California, June 1973.

7. Elspas, B.; K. N. Levitt; and R. J. Waldinger, "An interactive system for the verification of computer programs." SRI Project 1891, Stanford Research Institute, Menlo Park, California, Sept. 1973.

8. Elspas, B.; K. Levitt; R. Waldinger; and A. Waksman, "An assessment of techniques for proving program correctness." Computing Surveys, vol. 4,2 (June 1972), 97-147.

9. Floyd, R. W., "Assigning meanings to programs." Mathematical Aspects of Computer Science, J. T. Schwartz, ed., vol. 19 (American Mathematical Society, Providence, R. I. (1967)).

10. Green, M. W., "The use of difference equations as an aid to specifying assertions." "Research in interactive program proving techniques." SRI Report 8398-II, Stanford Research Institute, Menlo Park, California, May 1972.

11. Kaplan, D. M., "Regular expressions and the equivalence of programs." J. Comp. and Sys. Sciences 3, 4 (Nov. 1969), 361-386.

12. Katz, Shmuel M.; and Zohar Manna, "A heuristic approach to program verification." IJCAI-3 (Aug. 1973), 500-512.

13. King, J. C., "A program verifier." Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, Pa., 1969.

14. Luckham, D. C.; D. M. R. Park; and M. S. Patterson, "On formalized computer programs." J. Comp. and Sys. Sciences 4, 3 (June 1970), 220-249.

15. Manna, Zohar, "Properties of programs and the first-order predicate calculus." J. ACM 16, 2 (April 1970), 244-255.

REFERENCES (cont'd)

16. Mendelson, E., Introduction to mathematical logic. Van Nostrand Co., Princeton University Press, Princeton, N.J., 1964.

17. Moriconi, Mark, "An interactive program verification system." (tentative title), Ph.D. Dissertation, Computer Science Dept., University of Texas, Austin (under preparation).

18. Naur, P., "Proof of algorithms by general snapshots." BIT 6, 4 (1966), 310-316.

19. Pratt, Terrence W., "Kernel equivalence of programs and proving kernel equivalence and correctness by test cases." IJCAI-2 (Jan. 1971), 474-480.

20. Pratt, Terrence W., "Case descriptions of programs: an informal introduction." University of Texas C. S. Report TSN-32, Austin, Oct. 1972.

21. Shoenfield, J., Mathematical logic. Addison-Wesley Publishing Co., Inc., 1967.

22. Wegbreit, Ben, "The synthesis of loop predicates." Comm. ACM 17, 2 (Feb. 1974), 102-112.