STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO AIM-139
COMPUTER SCIENCE DEPARTMENT
REPORT NO. STAN-CS-71-189

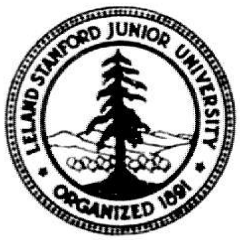
MATHEMATICAL THEORY OF PARTIAL CORRECTNESS

BY

ZOHAR MANNA

JANUARY 1971

COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY



MATHEMATICAL THEORY OF PARTIAL CORRECTNESS */

by

Zohar Manna

Computer Science Department

Stanford University

ABSTRACT: In this work we show that it is possible to express most properties regularly observed in algorithms in terms of 'partial correctness' (i.e., the property that the final results of the algorithm, if any, satisfy some given input-output relation).

This result is of special interest since 'partial correctness' has already been formulated in predicate calculus and in partial function logic for many classes of algorithms.

*/ The resesearch reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

A preliminary version of this work was presented under the title "Second-Order Mathematical Theory of Computation" at the ACM Symposium on Theory of Computing (May 1970).

Introduction

We normally distinguish between two classes of algorithms: deterministic algorithms and non-deterministic algorithms. A deterministic algorithm defines a single-valued (partial) function, while a non-deterministic algorithm defines a many-valued function. Therefore, while there are only a few properties of interest (mainly, termination, correctness, and equivalence) for deterministic algorithms, there are many more (determinacy, for example) for non-deterministic algorithms.

In this work, we show that it is possible to express most properties regularly observed in such algorithms in terms of the 'partial correctness' property (i.e., the property that the final results of the algorithm, if any, satisfy some given input-output relation).

This result is of special interest since 'partial correctness' has already been formulated in predicate calculus for many classes of deterministic algorithms, such as flowchart programs (Floyd (1967 a) and Manna (1969)), functional programs (Manna and Pnueli (1970)), and Algol-like programs (Ashcroft (1970)); and also for certain classes of non-deterministic algorithms, such as choice flowchart programs (Manna (1970)) and parallel flowchart programs (Ashcroft and Manna (1970)). See also Cooper (1969 a, 1969 b). Similarly, Manna and McCarthy (1970) have formulated 'partial correctness' of functional programs in partial function logic.

1. Deterministic Algorithms

An algorithm P (with input variable x and output variable z) is said to be deterministic if it defines a single-valued (partial) function $z = P(x)$ mapping D_x (the input domain) into D_z (the output domain). That is, for every $\xi \in D_x$, $P(\xi)$ is either undefined or defined with $P(\xi) \in D_z$.

Examples: In the sequel we shall discuss the following four deterministic algorithms for computing $z = x!$ where $D_x = D_z = \{\text{the non-negative integers}\}$.

(a) The flowchart programs P_1 (Figure 1) and P_2 (Figure 2). Here $(y_1, y_2) \leftarrow (y_1 - 1, y_1 \cdot y_2)$, for example, means that y_1 is replaced by $y_1 - 1$ and y_2 is replaced by $y_1 \cdot y_2$, simultaneously.

(b) The functional programs

P_3: $z = F(x)$ where

$F(y) \Leftarrow$ if $y = 0$ then 1 else $y \cdot F(y-1)$;

and

P_4: $z = F(x,0)$ where

$F(x,y) \Leftarrow$ if $y = x$ then 1 else $(y+1) \cdot F(x,y+1)$.

Here '$\Leftarrow$' stands for 'is defined recursively by' (see McCarthy (1963)).

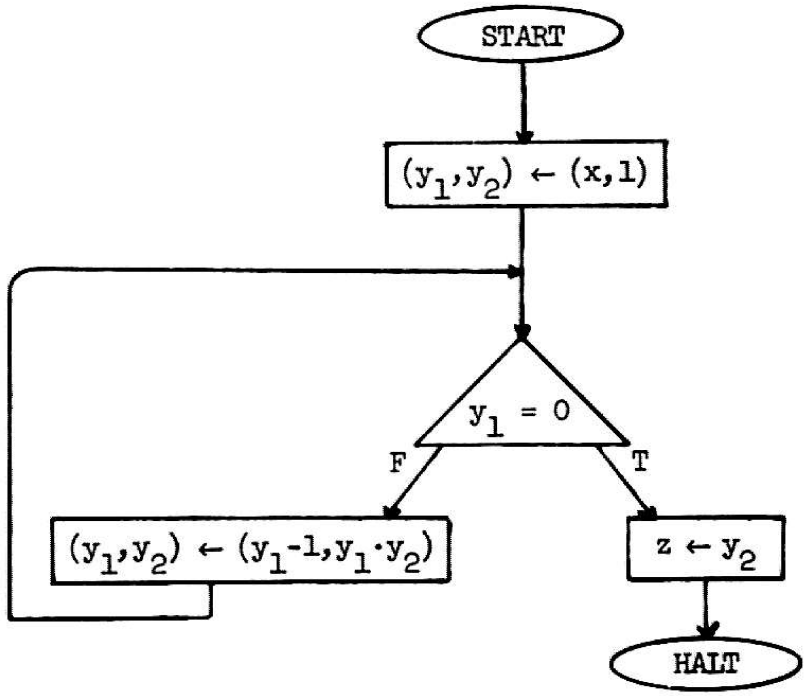


Figure 1: The flowchart program P_1 for computing $z = x!$

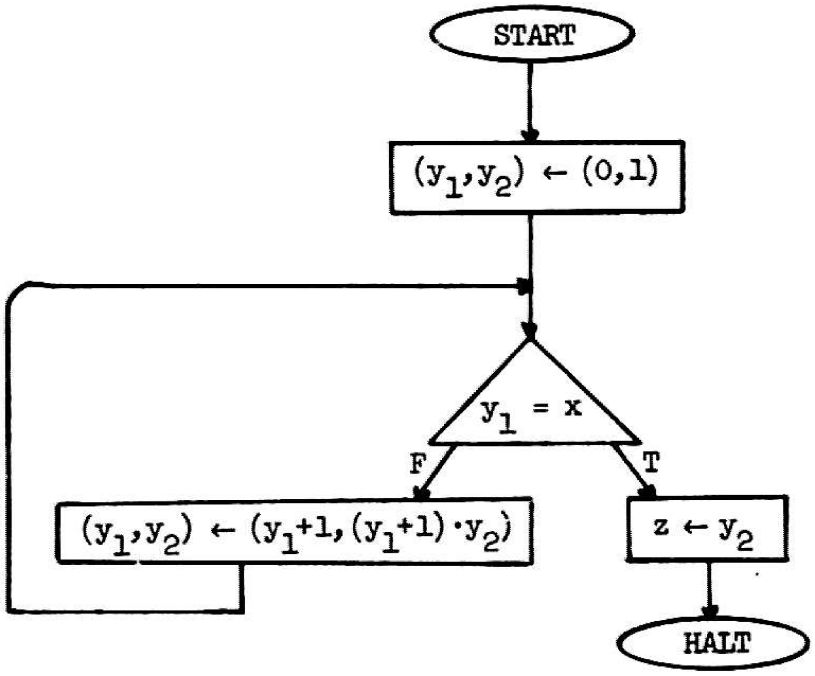


Figure 2: The flowchart program P_2 for computing $z = x!$

Let $\psi(x,z)$ be a total predicate over $D_x \times D_z$ (called the output predicate), and let $\xi \in D_x$. We say that

1. (i) (P,ξ) is partially correct with respect to ψ if either $P(\xi)$ is undefined, or $P(\xi)$ is defined and $\psi(\xi,P(\xi)) = T$;

   (ii) (P,ξ) is totally correct with respect to ψ if $P(\xi)$ is defined and $\psi(\xi,P(\xi)) = T$;

   (iii) (P,ξ) is defined if $P(\xi)$ is defined.

Let P_1 and P_2 be any two comparable deterministic algorithms, i.e., algorithms with the same input domain D_x and the same output domain D_z. We say that

2. (i) (P_1,ξ) and (P_2,ξ) are partially equivalent if either $P_1(\xi)$ or $P_2(\xi)$ is undefined, or both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) = P_2(\xi)$;

   (ii) (P_1,ξ) and (P_2,ξ) are totally equivalent if both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) = P_2(\xi)$.

3. (i) (P_1,ξ) is an extension of (P_2,ξ) if whenever $P_2(\xi)$ is defined, then so is $P_1(\xi)$ and $P_1(\xi) = P_2(\xi)$;

   (ii) (P_1,ξ) and (P_2,ξ) are equivalent if either both $P_1(\xi)$ and $P_2(\xi)$ are undefined, or both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) = P_2(\xi)$.

Our main purpose in this section is to show that all these properties can be expressed in terms of partial correctness as described in the following theorem.*/

---

*/ For abbreviation, we use $\sim\psi$ to define the predicate which is T exactly for those values where ψ is F , $\forall\psi$ to mean "for every output predicate ψ ...", and $\exists\psi$ to mean "there exists an output predicate ψ such that ..." .

THEOREM 1

(a) (P,ξ) is totally correct w.r.t. ψ <u>if and only if</u> (P,ξ) is not partially correct w.r.t. $\sim\psi$;

(b) (P,ξ) is defined <u>if and only if</u> (P,ξ) is not partially correct w.r.t. F (false);

(c) (P_1,ξ) is partially equivalent to (P_2,ξ) <u>if and only if</u> $\forall\psi$ [(P_1,ξ) is partially correct w.r.t. ψ or (P_2,ξ) is partially correct w.r.t. $\sim\psi$] ;

(d) (P_1,ξ) is totally equivalent to (P_2,ξ) <u>if and only if</u> $\forall\psi$ [(P_1,ξ) is not partially correct w.r.t. ψ or (P_2,ξ) is not partially correct w.r.t. $\sim\psi$] ;

(e) (P_1,ξ) is an extension of (P_2,ξ) <u>if and only if</u> $\forall\psi$ [(P_1,ξ) is partially correct w.r.t. ψ implies (P_2,ξ) is partially correct w.r.t. ψ] ; and finally

(f) (P_1,ξ) is equivalent to (P_2,ξ) <u>if and only if</u> $\forall\psi$ [(P_1,ξ) is partially correct w.r.t. ψ if and only if (P_2,ξ) is partially correct w.r.t. ψ] .

<u>Proof of Theorem 1</u>. The proof of (a) is straightforward. (b) is a special case of (a) since by definition (P_1,ξ) is defined if and only if it is totally correct w.r.t. T (true). (c), (d) and (e) are best proven by considering the corresponding contra-positive relations and using the fact that $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) \neq P_2(\xi)$ if and only if $P_1(\xi)$ and $P_2(\xi)$ are defined and $\exists\psi[\psi(\xi,P_1(\xi)) \neq \psi(\xi,P_2(\xi))]$.

(c') (P_1,ξ) is not partially equivalent to (P_2,ξ) (i.e., both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) \neq P_2(\xi)$) <u>if and only if</u> $\exists\psi$[(P_1,ξ) is not partially correct w.r.t. ψ and (P_2,ξ) is not partially correct w.r.t. $\sim\psi$] ;

(d') (P_1,ξ) is not toally equivalent to (P_2,ξ) (i.e., either $P_1(\xi)$ or $P_2(\xi)$ is undefined, or both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_i(\xi) \neq P_2(\xi)$) <u>if and only if</u> $\exists\Psi[(P_1,\xi)$ is partially correct w.r.t. Ψ and (P_2,ξ) is partially correct w.r.t. $\sim\Psi]$; and

(e') (P_1,ξ) is not an extension of (P_2,ξ) (i.e., either $P_2(\xi)$ is defined and $P_1(\xi)$ is undefined, or both $P_1(\xi)$ and $P_2(\xi)$ are defined and $P_1(\xi) \neq P_2(\xi)$) <u>if and only if</u> $\exists\Psi[(P_1,\xi)$ is partially correct w.r.t. Ψ and (P_2,ξ) is not partially correct w.r.t. $\Psi]$.

(f) follows directly from (e) since (P_1,ξ) is equivalent to (P_2,ξ) <u>if and only if</u> (P_1,ξ) is an extension of (P_2,ξ) and (P_2,ξ) is an extension of (P_1,ξ) .

Suppose for a given deterministic algorithm P (mapping integers into integers) we wish to formulate properties such as being total and monotonically increasing (i.e., $x > x' \Rightarrow P(x) > P(x')$). Unfortunately, our definitions of partial and total correctness are not general enough to include such simple properties in a natural way. However, we can include them by introducing more general notions of partial and total correctness.

Let P_i $(1 \leq i \leq n)$ be n deterministic algorithms with input variables x_i , output variables z_i , input domains D_{x_i} , and output domains D_{z_i} , respectively. Let $\tilde{\Psi}(x_1,z_1,\ldots,x_n,z_n)$ be any total predicate over $D_{x_1} \times D_{z_1} \times \ldots \times D_{x_n} \times D_{z_n}$ and let $\xi_i \in D_{x_i}$ $(1 \leq i \leq n)$. We say that

4. (i) $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are <u>partially correct</u> w.r.t. $\tilde{\Psi}$ if either at least one of the $P_i(\xi_i)$ is undefined, or each $P_i(\xi_i)$ is defined and $\tilde{\Psi}(\xi_1,P_1(\xi_1),\ldots,\xi_n,P_n(\xi_n)) = T$.

(ii) $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are <u>totally correct</u> w.r.t. $\tilde{\Psi}$ if each $P_i(\xi_i)$ is defined and $\tilde{\Psi}(\xi_1,P_1(\xi_1),\ldots,\xi_n,P_n(\xi_n)) = T$.

Note that for $n = 1$ we obtain properties 1(i) and 1(ii) as special cases of properties 4(i) and 4(ii), respectively. For $n = 2$ and $\tilde{\psi}(x_1,z_1,x_2,z_2)$: $x_1 = x_2 \supset z_1 = z_2$, properties 4(i) and 4(ii) reflect properties 2(i) and 2(ii), respectively. For $n = 2$ and $\tilde{\psi}(x_1,z_1,x_2,z_2)$: $x_1 > x_2 \supset z_1 > z_2$ where P_1 and P_2 are identical to P, we obtain the above monotonicity property.

It is interesting that these general notions of correctness can still be expressed just by means of the usual partial correctness, as described below.

THEOREM 2

(a) $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are partially correct w.r.t. $\tilde{\psi}$ if and only if

$\exists\psi_1\ldots\exists\psi_n\{$ $P_1(\xi_1)$ is partially correct w.r.t. ψ_1
and $P_2(\xi_2)$ is partially correct w.r.t. ψ_2
$\vdots$
and $P_n(\xi_n)$ is partially correct w.r.t. ψ_n
and $\forall y_1\ldots\forall y_n[\psi_1(\xi_1,y_1)$ and ... and $\psi_n(\xi_n,y_n)$ implies $\tilde{\psi}(\xi_1,y_1,\ldots,\xi_n,y_n)]\}$;

(b) $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are totally correct w.r.t. $\tilde{\psi}$ if and only if

$\forall\psi_1\ldots\forall\psi_n\{$ $P_1(\xi_1)$ is partially correct w.r.t. ψ_1
and $P_2(\xi_2)$ is partially correct w.r.t. ψ_2
$\vdots$
and $P_n(\xi_n)$ is partially correct w.r.t. ψ_n
implies $\exists y_1\ldots\exists y_n[\psi_1(\xi_1,y_1)$ and ... and $\psi_n(\xi_n,y_n)$ and $\tilde{\psi}(\xi_1,y_1,\ldots,\xi_n,y_n)]\}$.

Proof of Theorem 2. It is straightforward that the right-hand side of (a) implies the left-hand side. To prove that the left-hand side implies the right-hand side, choose ψ_i in such a way that $\psi_i(\xi_i,\eta_i) = T$ if and only if $P_i(\xi_i)$ is defined and $\eta_i = P_i(\xi_i)$. (b) follows from (a) since $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are totally correct w.r.t. $\tilde{\psi}$ if and only if $(P_1,\xi_1),\ldots,(P_n,\xi_n)$ are not partially correct w.r.t. $\sim\tilde{\psi}$.

2. Formulation of Partial Correctness of Deterministic Algorithms

The above results imply that if one knows, for example, how to formulate partial correctness of a given deterministic algorithm in predicate calculus, the formulation of many other properties of the algorithm in predicate calculus is straightforward. As a matter of fact, partial correctness has already been formulated in predicate calculus for many classes of deterministic algorithms.

In this section we illustrate the flavor of such formulations.

(A) Flowchart Programs and Predicate Calculus

Let us consider, for example, a flowchart program P of the form described in Figure 3, with a given output predicate $\psi(x,z)$ over $D_x \times D_z$. Here, input(x) maps D_x into D_y, test(x,y) is a predicate over $D_x \times D_y$, operator(x,y) maps $D_x \times D_y$ into D_y, and output(x,y) maps $D_x \times D_y$ into D_z.

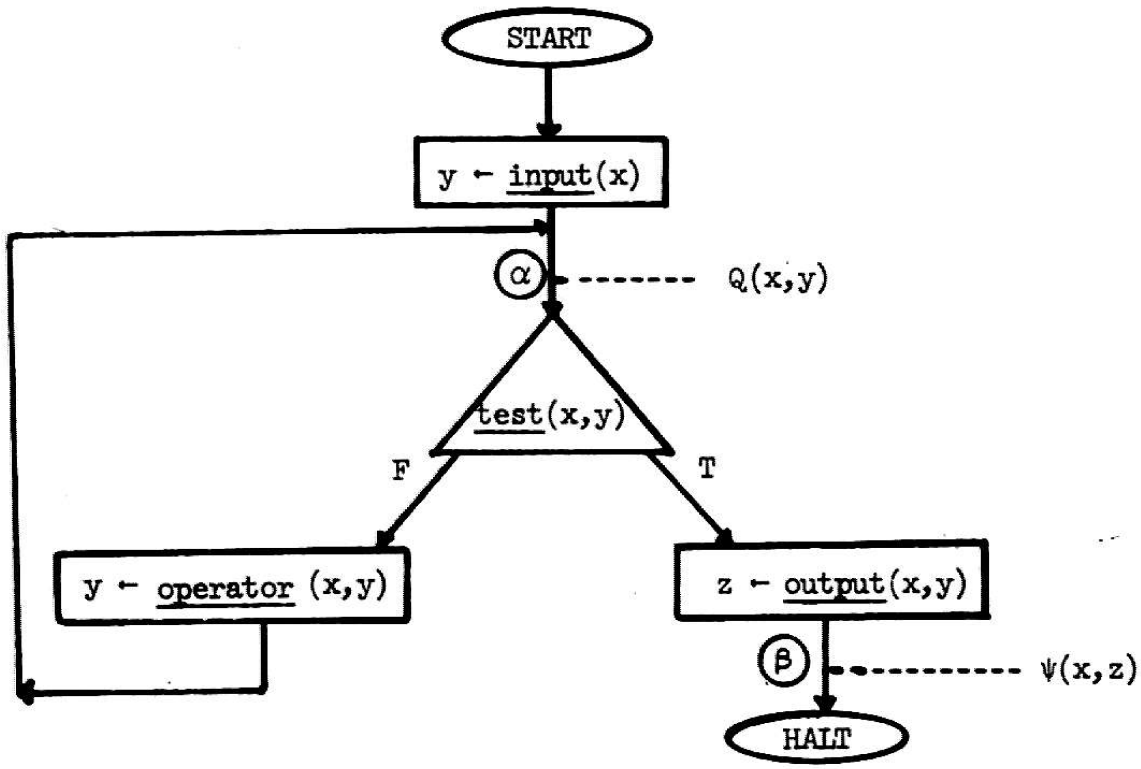


Figure 3: The flowchart program P

We associate a predicate variable (unspecified induction hypothesis) $Q(x,y)$ with arc α and the given output predicate $\psi(x,z)$ with arc β, and construct the following formula $W_P(x,\psi)$:

$$\exists Q\{\ Q(x,\underline{input}(x)) \qquad \text{--- initialization}$$
$$\wedge\ \forall y[Q(x,y) \wedge \sim \underline{test}(x,y) \supset Q(x,\underline{operator}(x,y))] \qquad \text{--- induction}$$
$$\wedge\ \forall y[Q(x,y) \wedge \underline{test}(x,y) \supset \psi(x,\underline{output}(x,y))]\} \qquad \text{--- conclusion}$$

or equivalently,

$$\exists Q\{\ Q(x,\underline{input}(x)) \qquad \text{--- initialization}$$
$$\wedge\ \forall y[Q(x,y) \supset \text{IF } \underline{test}(x,y) \text{ THEN } \psi(x,\underline{output}(x,y)) \qquad \text{--- conclusion}$$
$$\text{ELSE } Q(x,\underline{operator}(x,y))]\}\ . \qquad \text{--- induction}$$

Here, IF A THEN B ELSE C stands for $(A \supset B) \wedge (\sim A \supset C)$. Note that $D \supset$ IF A THEN B ELSE C is logically equivalent to $(D \wedge A \supset B) \wedge (D \wedge \sim A \supset C)$.

The key result is that for any given input $\xi \in D_x$, (P,ξ) is partially correct w.r.t. ψ <u>if and only if</u> $W_P(\xi,\psi)$ is true (Manna (1969)).

<u>Example 1</u>: In particular, for the flowchart program P_1 (Figure 1), it follows that:*/ (P_1,ξ) is partially correct w.r.t. $z = x!$ if and only if $W_{P_1}(\xi, z = x!)$ is true, where $W_{P_1}(\xi, z = x!)$ is

$$\exists Q\{\ Q(\xi,\xi,1)$$
$$\wedge\ \forall y_1 \forall y_2[Q(\xi,y_1,y_2) \supset \text{IF } y_1 = 0 \text{ THEN } y_2 = \xi! \text{ ELSE } Q(\xi,y_1-1,y_1 \cdot y_2)]\}\ .$$

Note that for $Q(\xi,y_1,y_2)$ being the predicate $y_2 \cdot y_1! = \xi!$, the formula in braces { } is true.

---

*/ Here, $D_x = D_z =$ {the non-negative integers} , $y = (y_1,y_2)$, and $D_y =$ {all pairs of non-negative integers} .

Example 2: For the flowchart program P_2 (Figure 2), it follows similarly that: (P_2,ξ) is partially correct w.r.t. $z = x!$ if and only if $W_{P_2}(\xi,z=x!)$ is true, where $W_{P_2}(\xi,z=x!)$ is

$$\exists Q\{\ Q(\xi,0,1)$$
$$\wedge\ \forall y_1\forall y_2[Q(\xi,y_1,y_2) \supset \text{IF } y_1 = \xi \text{ THEN } y_2 = \xi! \text{ ELSE } Q(\xi,y_1+1,(y_1+1)\cdot y_2)]\}\ .$$

Note that for $Q(\xi,y_1,y_2)$ being the predicate $y_2 = y_1!$, the formula in braces $\{\ \}$ is true.

(B) Functional Programs and Predicate Calculus

Consider, for example, a functional program P of the form

$$z = F(x,\underline{input}(x)) \text{ where}$$
$$F(x,y) \Leftarrow \underline{if}\ \underline{test}(x,y)\ \underline{then}\ \underline{output}(x,y)$$
$$\underline{else}\ \underline{operator1}(x,y,F(x,\underline{operator2}(x,y)))\ ,$$

with a given output predicate $\psi(x,z)$ over $D_x \times D_z$. Here, $\underline{input}(x)$ maps D_x into D_y, $\underline{test}(x,y)$ is a predicate over $D_x \times D_y$, $\underline{output}(x,y)$ maps $D_x \times D_y$ into D_z, $\underline{operator1}$ maps $D_x \times D_y \times D_z$ into D_z, and $\underline{operator2}$ maps $D_x \times D_y$ into D_y.

We associate a predicate variable (unspecified induction hypothesis) $Q(x,y,z)$ with $F(x,y)$, and construct the following formula $W_P(x,\psi)$:

$$\exists Q\{\ \forall z[Q(x,\underline{input}(x),z) \supset \psi(x,z)] \qquad \text{-- conclusion}$$
$$\wedge\ \forall y[\text{IF } \underline{test}(x,y) \text{ THEN } Q(x,y,\underline{output}(x,y)) \qquad \text{-- initialization}$$
$$\text{ELSE } \forall t[Q(x,\underline{operator2}(x,y),t)$$
$$\supset Q(x,y,\underline{operator1}(x,y,t))]]\} \text{ -- induction}$$

The key result is that for any given input $\xi \in D_x$, (P,ξ) is partially correct w.r.t. ψ *if and only if* $W_P(\xi,\psi)$ is true (Manna and Pnueli (1970), see also Park (1970)).

Example 3: For the functional program P_3 :

$z = F(x)$ where

$F(y) \Leftarrow$ if $y = 0$ then 1 else $y \cdot F(y-1)$,

it follows that: (P_3, ξ) is partially correct w.r.t. $z = x!$ if and only if $W_{P_3}(\xi, z = x!)$ is true, where $W_{P_3}(\xi, z = x!)$ is

$$\exists Q\{\ \forall z[Q(\xi, z) \supset z = \xi!]$$
$$\wedge\ \forall y[\text{IF } y = 0 \text{ THEN } Q(y,1) \text{ ELSE } \forall t[Q(y-1,t) \supset Q(y, y \cdot t)]]\} .$$

Note that for $Q(y,z)$ being the predicate $z = y!$ the formula in braces { } is true.

Example 4: For the functional program P_4 :

$z = F(x,0)$ where

$F(x,y) \Leftarrow$ if $y = x$ then 1 else $(y+1) \cdot F(x, y+1)$,

it follows that: (P_4, ξ) is partially correct w.r.t. $z = x!$ if and only if $W_{P_4}(\xi, z = x!)$ is true, where $W_{P_4}(\xi, z = x!)$ is

$$\exists Q\{\ \forall z[Q(\xi, 0, z) \supset z = \xi!]$$
$$\wedge\ \forall y[\text{IF } y = \xi \text{ THEN } Q(\xi, y, 1) \text{ ELSE } \forall t[Q(\xi, y+1, t) \supset Q(\xi, y, (y+1) \cdot t)]]\} .$$

Note that for $Q(\xi, y, z)$ being the predicate $z \cdot y! = \xi!$, the formula in braces { } is true.

The formulas constructed here are independent of the syntax of the language in which the algorithms are expressed, and, therefore, we can use our results to formulate in predicate calculus the equivalence of algorithms defined by different languages. From part (f) of Theorem 1 it follows, for example, that for every input ξ , (P_1, ξ) and (P_3, ξ) are equivalent if and only if $\forall \psi[W_{P_1}(\xi, \psi) \equiv W_{P_3}(\xi, \psi)]$ is true.

The reader should realize that the flowchart program P (Figure 3) can be represented equivalently (see McCarthy (1962)) by the functional program P' :

$$z = F(x, \underline{input}(x)) \text{ where}$$
$$F(x,y) \Leftarrow \underline{if}\ \underline{test}(x,y)\ \underline{then}\ \underline{output}(x,y)\ \underline{else}\ F(x, \underline{operator}(x,y)) \ .$$

However, $W_{P'}(x,\psi)$ is

$$\exists Q\{\ \forall z[Q(x, \underline{input}(x), z) \supset \psi(x,z)]$$
$$\wedge\ \forall y[\text{IF } \underline{test}(x,y) \text{ THEN } Q(x,y, \underline{output}(x,y))$$
$$\text{ELSE } \forall t[Q(x, \underline{operator}(x,y), t) \supset Q(x,y,t)]]\} \ ;$$

while $W_P(x,\psi)$ was

$$\exists Q\{\ Q(x, \underline{input}(x))$$
$$\wedge\ \forall y[Q(x,y) \supset \text{IF } \underline{test}(x,y) \text{ THEN } \psi(x, \underline{output}(x,y)) \text{ ELSE } Q(x, \underline{operator}(x,y))]\} \ .$$

Although both $W_P(x,\psi)$ and $W_{P'}(x,\psi)$ essentially formulate partial correctness of (P,x) w.r.t. ψ, they seem to be quite different. Intuitively, the difference between the two formulations is that $Q(x,y)$ in $W_P(x,\psi)$ represents all current values of (x,y) at arc α during the computation of P, while $Q(x,y,z)$ in $W_{P'}(x,\psi)$ represents the final value of z when computation of P starts at arc α with initial values (x,y) .

(C) Functional Programs and Partial Function Logic

Consider again a functional program P of the form

$$z = F(x, \underline{input}(x)) \text{ where}$$
$$F(x,y) \Leftarrow \underline{if}\ \underline{test}(x,y)\ \underline{then}\ \underline{output}(x,y)$$
$$\underline{else}\ \underline{operator1}(x,y,F(x, \underline{operator2}(x,y))) \ ,$$

with a given output predicate $\psi(x,z)$.

We construct the following formula $\tilde{W}_P(x,\psi)$:

$$\exists F\{\ [{}^*F(x,\underline{input}(x)) \supset \psi(x,F(x,\underline{input}(x)))]$$
$$\wedge\ \forall y[F(x,y) \overset{*}{=} \underline{if}\ \underline{test}(x,y)\ \underline{then}\ \underline{output}(x,y)$$
$$\underline{else}\ \underline{operator1}(x,y,F(x,\underline{operator2}(x,y)))]\}\ .$$

Here, "$\exists F$" stands for "there exists a partial function F mapping $D_x \times D_y$ into D_z such that ..."; "${}^*F(x,\underline{input}(x))$" stands for the total predicate (mapping D_x into $\{T,F\}$) "$F(x,\underline{input}(x))$ is defined"; and $\overset{*}{=}$ is just the natural extension of the usual equality relation, defined as follows: $A \overset{*}{=} B$ if and only if either both expressions A and B are defined and represent the same element (of D_z, in this case) or both expressions are undefined.

The key result is that for every given $\xi \in D_x$, (P,ξ) is partially correct w.r.t. ψ <u>if and only if</u> $\tilde{W}_P(\xi,\psi)$ is true (Manna and McCarthy (1970)).

<u>Example 5</u>: For the functional program P_4 :

$$z = F(x,0)\ \underline{where}$$
$$F(x,y) \Leftarrow \underline{if}\ y = x\ \underline{then}\ 1\ \underline{else}\ (y+1)\cdot F(x,y+1)\ ,$$

it follows that: (P_4,ξ) is partially correct w.r.t. $z = x!$ if and only if $\tilde{W}_{P_4}(\xi\ ,\ z = x!)$ is true, where $\tilde{W}_{P_4}(\xi\ ,\ z = x!)$ is

$$\exists F\{\ [{}^*F(\xi,0) \supset F(\xi,0) = \xi!]$$
$$\wedge\ \forall y[F(\xi,y) \overset{*}{=} \underline{if}\ y = \xi\ \underline{then}\ 1\ \underline{else}\ (y+1)\cdot F(\xi,y+1)]\}\ .$$

Note that for $F(\xi,y)$ being the partial function

$$F(\xi,y) = \begin{cases} \xi!/y! & \text{if } y \leq \xi \\ \text{undefined} & \text{if } y > \xi \end{cases}$$

the formula in braces { } is true.

3. Non-Deterministic Algorithms

One natural extension of our study is obtained by considering non-deterministic algorithms rather than deterministic algorithms.

An algorithm P (with input variable x and output variable z) is said to be non-deterministic if it defines a many-valued function $P(x)$, mapping elements of D_x (the input domain) into subsets of D_z (the output domain); that is, for every $\xi \in D_x$, $P(\xi)$ is a (possibly empty) subset Z of D_z, where each $\zeta \in Z$ is the final result of some computation of P with input ξ.

Examples: We first describe three non-deterministic programs for computing $z = x!$, making use of the deterministic programs P_1-P_4 introduced in Section 1.

(a) Parallel flowchart program: In Figure 4 we have described a simple parallel flowchart program P_5 for computing $z = x!$. The program includes a 'BEGIN-END' block which consists of two branches, the left branch being the body of program P_1 and the right branch being the body of program P_2, after changing the test statements to $y_1 = y_1'$ in both.

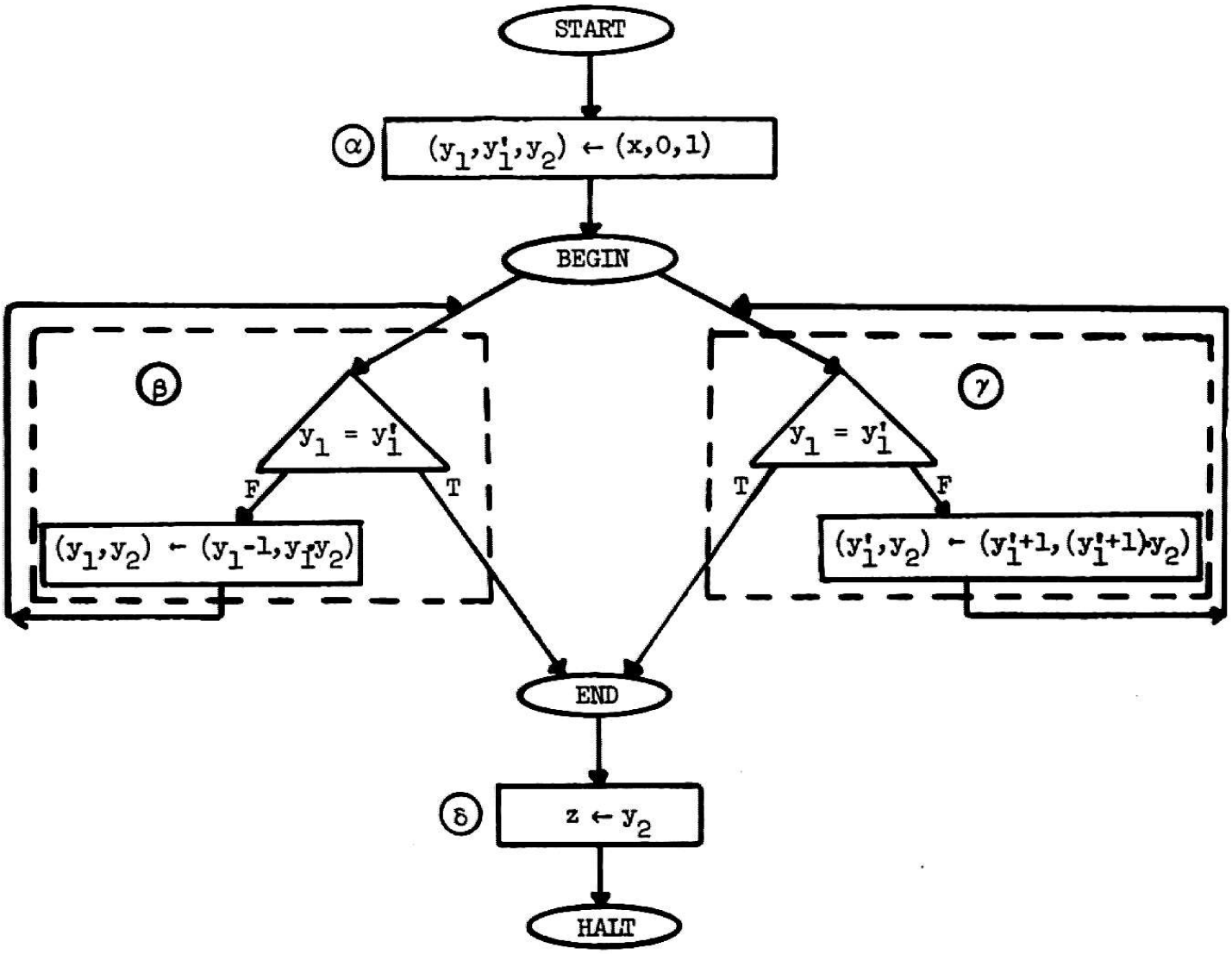


Figure 4: The parallel flowchart program P_5 for computing $z = x!$

The program is executed as follows. First statement α is executed. Entering the block either the statements in β or the statements in γ are executed, chosen arbitrarily. The execution proceeds asynchronously, i.e., between the execution of two consecutive β's , we may execute an arbitrary number of γ's ; and conversely, between the execution of two consecutive γ's we may execute an arbitrary number of β's . β and γ cannot be executed at the same time. Therefore, one can consider execution to be performed with a single processor switching between the two branches. We exit from the block and execute statement δ when either of the two branches reaches the END node. Such parallel programs are discussed in detail in Ashcroft and Manna (1970).

(b) Choice flowchart program: In Figure 5 we have described a choice flowchart program for computing $z = x!$. A branch of the form  is called a choice branch. It means that upon reaching the choice branch during execution of the program, we are allowed to proceed with either branch, chosen arbitrarily. Such choice flowchart programs have been discussed in detail by Floyd (1967 b).

Note that for any given input x both P_5 and P_6 yield the same set of computations. For $x = 3$, for example, there are exactly 8 different possible executions of each program. In general, for every non-negative input x , there are 2^x different possible computations of each program.

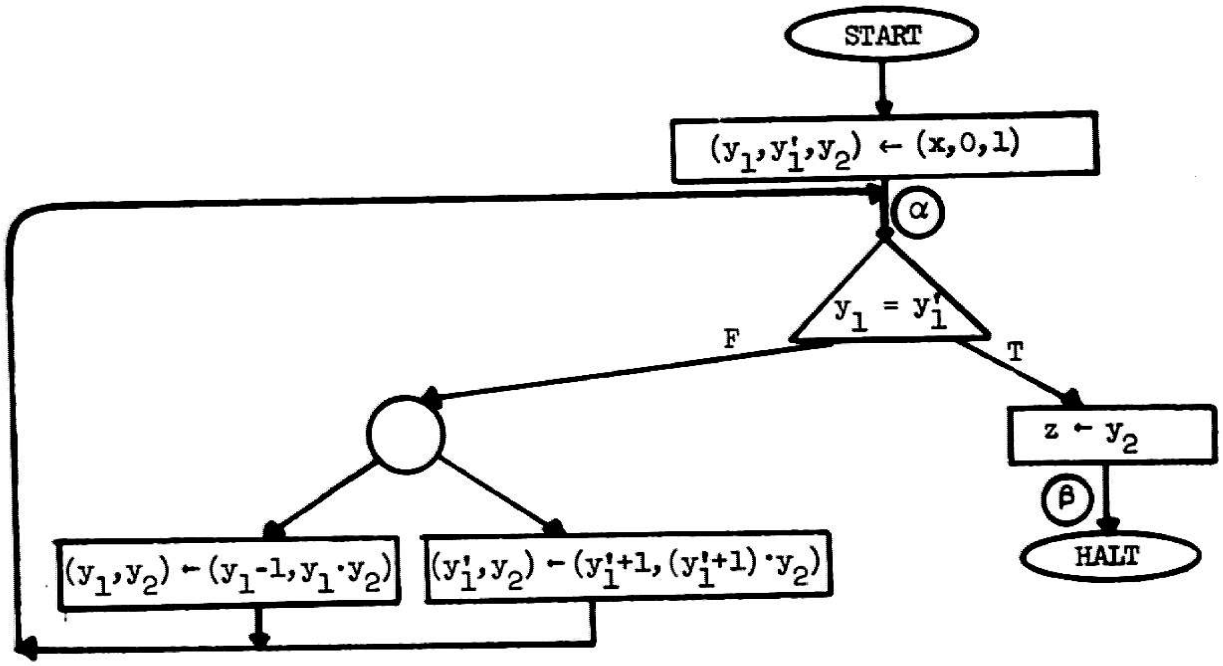


Figure 5: The choice flowchart program P_6 for computing $z = x!$

(c) Choice functional program: Consider the following choice functional program P_7 :

$z = F(x,0)$ where

$F(y,y') \Leftarrow$ if $y = y'$ then 1 else choice$(y \cdot F(y-1,y'), (y'+1) \cdot F(y,y'+1))$.

The choice function here has the same meaning as the choice branch in P_6 ; it corresponds to McCarthy's (1963) amb (ambiguous) function. For every non-negative input x there are again 2^x different possible computations of P_7 .

In this section we shall discuss several properties of non-deterministic algorithms. For non-deterministic algorithm P and input $\xi \in D_x$ we say that

1.(i) (P,ξ) is $\exists$-defined if there exists a finite computation P with input ξ (or, equivalently, $P(\xi) \neq \emptyset$);

(ii) (P,ξ) is $\forall$-defined if every computation of P with input ξ is finite;

(iii) (P,ξ) is partially determinate if all finite computations of P with input ξ yield the same final result (or, equivalently, $P(\xi)$ is either empty or a singleton);

(iv) (P,ξ) is totally determinate if all computations of P with input ξ are finite and yield the same final result.

Let $\psi(x,z)$ be a total predicate over $D_x \times D_z$, and let $\xi \in D_x$. A finite computation of P with input ξ is said to be correct w.r.t. ψ if for its final value ζ, $\psi(\xi,\zeta) = T$. We say that

2.(i) (P,ξ) is partially $\exists$-correct w.r.t. ψ if either there exists an infinite computation of P with input ξ, or there exists a finite computation of P with input ξ which is correct w.r.t. ψ;

(ii) (P,ξ) is totally $\exists$-correct w.r.t. ψ if there exists a finite computation of P with input ξ which is correct w.r.t. ψ;

(iii) (P,ξ) is partially $\forall$-correct w.r.t. ψ if every finite computation of P with input ξ is correct w.r.t. ψ;

(iv) (P,ξ) is totally $\forall$-correct w.r.t. ψ if every computation of P with input ξ is finite and is correct w.r.t. ψ.

Let P_1 and P_2 be any two comparable non-deterministic algorithms, i.e., algorithms with the same input domain D_x and the same output domain D_z We say that

3.(i) (P_1,ξ) and (P_2,ξ) are <u>partially determinate-equivalent</u> if all finite computations of P_1 and P_2 with input ξ yield the same final result (or, equivalently, $P_1(\xi) \cup P_2(\xi)$ is either empty or a singleton).

(ii) (P_1,ξ) and (P_2,ξ) are <u>totally determinate-equivalent</u> if all computations of P_1 and P_2 with input ξ are finite and yield the same final result.

4.(i) (P_1,ξ) <u>partially extends</u> (P_2,ξ) if, for every finite computation of P_2 with input ξ, there exists a finite computation of P_1 with input ξ that yields the same final value (or, equivalently, $P_1(\xi) \supseteq P_2(\xi)$);

(ii) (P_1,ξ) <u>totally extends</u> (P_2,ξ) if (P_1,ξ) partially extends (P_2,ξ), and if there exists an infinite computation of P_2 with input ξ, then there is also an infinite computation of P_1 with input ξ.

5.(i) (P_1,ξ) and (P_2,ξ) are <u>partially equivalent</u> if (P_1,ξ) partially extends (P_2,ξ) and conversely (or, equivalently, $P_1(\xi) = P_2(\xi)$);

(ii) (P_1,ξ) and (P_2,ξ) are <u>totally equivalent</u> if (P_1,ξ) totally extends (P_2,ξ) and conversely.

Our main purpose in this section is to show that all these properties can be expressed in terms of the two notions of partial correctness, namely partial $\exists$-correctness and partial $\forall$-correctness.

THEOREM 3

(a) (P,ξ) is $\exists$-defined <u>if and only if</u> (P,ξ) is not partially $\forall$-correct w.r.t. F (false);

(b) (P,ξ) is $\forall$-defined <u>if and only if</u> (P,ξ) is not partially $\exists$-correct w.r.t. F (false);

(c) (P,ξ) is partially determinate <u>if and only if</u> $\forall\psi[(P,\xi)$ is partially $\forall$-correct w.r.t. ψ or (P,ξ) is partially $\forall$-correct w.r.t. $\sim\psi]$;

(d) (P,ξ) is totally determinate <u>if and only if</u> $\forall\psi[(P,\xi)$ is not partially $\exists$-correct w.r.t. ψ or (P,ξ) is not partially $\exists$-correct w.r.t. $\sim\psi]$;

(e) (P,ξ) is totally $\exists$-correct w.r.t. ψ <u>if and only if</u> (P,ξ) is not partially $\forall$-correct w.r.t. $\sim\psi$;

(f) (P,ξ) is totally $\forall$-correct w.r.t. ψ <u>if and only if</u> (P,ξ) is not partially $\exists$-correct w.r.t. $\sim\psi$;

(g) (P_1,ξ) and (P_2,ξ) are partially determinate-equivalent <u>if and only if</u> $\forall\psi[(P_1,\xi)$ is partially $\forall$-correct w.r.t. ψ or (P_2,ξ) is partially $\forall$-correct w.r.t. $\sim\psi]$;

(h) (P_1,ξ) and (P_2,ξ) are totally determinate-equivalent <u>if and only if</u> $\forall\psi[(P_1,\xi)$ is not partially $\exists$-correct w.r.t. ψ or (P_2,ξ) is not partially $\exists$-correct w.r.t. $\sim\psi]$;

(i) (P_1,ξ) partially extends (P_2,ξ) <u>if and only if</u> $\forall\psi[(P_1,\xi)$ is partially $\forall$-correct w.r.t. ψ implies (P_2,ξ) is partially $\forall$-correct w.r.t. $\psi]$;

(j) (P_1,ξ) totally extends (P_2,ξ) <u>if and only if</u> $\forall\psi[(P_2,\xi)$ is partially $\exists$-correct w.r.t. ψ implies (P_1,ξ) is partially $\exists$-correct w.r.t. $\psi]$;

(k) (P_1,ξ) and (P_2,ξ) are partially equivalent <u>if and only if</u> $\forall\psi[(P_1,\xi)$ is partially $\forall$-correct w.r.t. ψ if and only if (P_2,ξ) is partially $\forall$-correct w.r.t. $\psi]$;

(ℓ) (P_1,ξ) and (P_2,ξ) are totally equivalent <u>if and only if</u> $\forall\psi[(P_1,\xi)$ is partially $\exists$-correct w.r.t. ψ if and only if (P_2,ξ) is partially $\exists$-correct w.r.t. $\psi]$.

<u>Proof of Theorem 3</u>: (a), (b), (e) and (f) are straightforward by definition. (c), (d), (g), (h), (i), and (j) are best proved by considering the corresponding contra-positive relations. (k) and (ℓ) follows from (i) and (j), respectively.

4. Formulation of Partial Correctness of Non-Deterministic Algorithms

For a given non-deterministic program P and an output predicate $\psi(x,z)$, we would like to construct two formulas $W^{\exists}(x,\psi)$ and $W^{\forall}(x,\psi)$ in predicate calculus, such that for every given input value $\xi\in D_x$:

(i) (P,ξ) is partially $\exists$-correct w.r.t. ψ if and only if $W^{\exists}(\xi,\psi)$ is true, and

(ii) (P,ξ) is partially $\forall$-correct w.r.t. ψ if and only if $W^{\forall}(\xi,\psi)$ is true.

Then, using the formulas $W^{\exists}(x,\psi)$ and $W^{\forall}(x,\psi)$, the formulation of the other properties of P in predicate calculus is straightforward.

Following Ashcroft and Manna (1970), one can formulate properties of the parallel flowchart P_5 by first translating it to the equivalent choice flowchart program P_6 and then make use of the formulas $W^{\exists}_{P_6}(x,\psi)$ and $W^{\forall}_{P_6}(x,\psi)$. We shall therefore illustrate the construction of $W^{\exists}(x,\psi)$ and

$W^{\forall}(x,\psi)$ only for the choice flowchart program P_6 (Figure 5) and the choice functional program P_7. The main idea behind this formulation is that the effect of the choice branch is represented by an '$\vee$' connective in $W^{\exists}(x,\psi)$, while it is represented by an '$\wedge$' connective in $W^{\forall}(x,\psi)$ (see Manna (1970)).

To construct $W^{\forall}_{P_6}(\xi, z = x!)$, associate the predicate variable $Q(\xi,y_1,y_1',y_2)$ with arc α in Figure 5 and the predicate variable $z = x!$ with arc β. Then $W^{\forall}_{P_6}(\xi, z = x!)$ is

$$\exists Q\{\ Q(\xi,\xi,0,1)$$
$$\wedge\ \forall y_1 \forall y_1' \forall y_2 [Q(\xi,y_1,y_1',y_2) \supset \text{IF } y_1 = y_1' \text{ THEN } y_2 = \xi!$$
$$\text{ELSE } [Q(\xi,y_1-1,y_1',y_1 \cdot y_2) \wedge Q(\xi,y_1,y_1'+1,(y_1'+1)\cdot y_2)]]\} .$$

The reader can verify easily that for every non-negative integer ξ, the formula $W^{\forall}_{P_6}(\xi, z = x!)$ is true for $Q(\xi,y_1,y_1',y_2)$ being the predicate $y_2 \cdot y_1! = \xi! \cdot y_1'!$. $W^{\exists}_{P_6}(\xi, z = x!)$ is similar with the '$\wedge$' connective replaced by '$\vee$'.

To construct $W^{\forall}_{P_7}(\xi, z = x!)$, associate the predicate variable $Q(y,y',z)$ with the function variable $F(y,y')$. Then $W^{\forall}_{P_7}(\xi, z = x!)$ is:

$$\exists Q\{\ \forall z[Q(\xi,0,z) \supset z = \xi!]$$
$$\wedge\ \forall y \forall y' [\text{IF } y = y' \text{ THEN } Q(y,y',1)$$
$$\text{ELSE } \forall t[Q(y-1,y',t) \supset Q(y,y',y\cdot t)]$$
$$\wedge\ \forall t[Q(y,y'+1,t) \supset Q(y,y',(y'+1)\cdot t)]]\} .$$

The reader can verify easily that for every non-negative integer ξ, the formula $W^{\forall}_{P_7}(\xi, z = x!)$ is true for $Q(y,y',z)$ being the predicate $z \cdot y'! = y!$. $W^{\exists}_{P_7}(\xi, z = x!)$ is similar with the '$\wedge$' connective replaced by '$\vee$'.

Acknowledgments: I am indebted to Edward Ashcroft and Stephen Ness for many stimulating discussions and also for their critical reading of the manuscript and subsequent helpful suggestions.

References

E. A. ASHCROFT (1970), "Mathematical Logic Applied to the Semantics of Computer Programs," Ph.D. Thesis, Imperial College, London.

E. A. ASHCROFT and Z. MANNA (1970), "Formalization of Properties of Parallel Programs," in Machine Intelligence 6 (Ed. Meltzer and Michie), Edinburgh University Press.

D. C. COOPER (1969 a), "Program Scheme Equivalences and Second-Order Logic," in Machine Intelligence 4 (Eds. Meltzer and Michie), Edinburgh University Press, 3-15.

D. C. COOPER (1969 b), "Program Schemes, Programs and Logic," Computation Services Department, University College of Swansea, Memo No. 6.

R. W. FLOYD (1967 a), "Assigning Meaning to Programs," in Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Vol. 19, 19-32.

R. W. FLOYD (1967 b), "Non-deterministic Algorithms," JACM (October 1967).

Z. MANNA (1969), "The Correctness of Programs," J. of Computer and System Sciences, Vol. 3, No. 2.

Z. MANNA (1970), "The Correctness of Non-deterministic Programs," Artificial Intelligence J., Vol. 1, No. 1.

Z. MANNA and J. McCARTHY (1970), "Properties of Programs and Partial Function Logic," in Machine Intelligence 5 (Eds. Meltzer and Michie), Edinburgh University Press, 79-98.

Z. MANNA and A. PNUELI (1970), "Formalization of Properties of Functional Programs," JACM, Vol. 17, No. 3.

J. McCARTHY (1962), "Towards a Mathematical Science of Computation," Proc. IFIP Congress 62, North-Holland, Amsterdam.

J. McCARTHY (1963), "A Basis for a Mathematical Theory of Computation," in Computer Programming and Formal Systems (Eds. Braffort and Hirshberg), North Holland, Amsterdam.

D. PARK (1970), "Fixpoint Induction and Proofs of Program Properties," in Machine Intelligence 5 (Eds. Meltzer and Michie), Edinburgh University Press, 59-78.