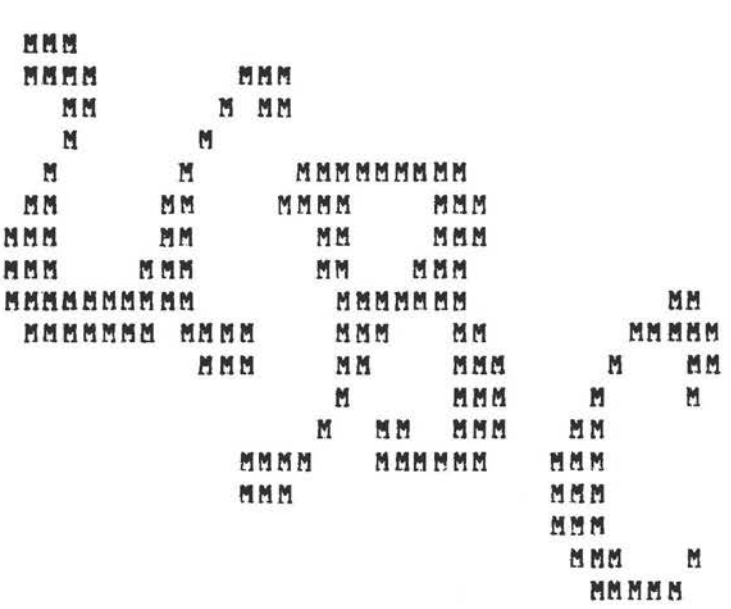


The Logical Semantics of Program Schemas and Program Verification

by

Akira Kanda

Technical Report 76-11

December 1976

Department of Computer Science
University of British Columbia
Vancouver, B. C.

The Logical Semantics of Program Schemas and Program Verification

Akira Kanda
Department of Computer Science
University of British Columbia
Vancouver 8, B.C.
Canada

ABSTRACT

Most work in the fixed-point semantics of programs has been based on a mathematical approach in which we specify the fixed-points mathematically. The only exception is the work by Park which takes a logical approach vis Tarski type fixed-point semantics. We reevaluate this logical approach in this paper. We will develop a logical approach via Scott type fixed-point semantics. Very powerful and easy to use induction methods will be obtained. It will be observed that several ad-hoc but powerful program verification methods are special cases of these induction methods. Also an interesting logical semantic formulation of terminating program schemas will be given. The computational aspect of the Scott type of logical semantics will not be emphasized in this paper, but it might provide very important and helpful information for finding inductive assertions.

ACKNOWLEDGEMENT

The auther wishes to thank prof. R. Reiter and prof. D.M. Symes for helpful discussions and encouragement. This research was partially supported by National Research Council of Canada through grant A7642.

1. Introduction

Several significant attempts have been made to define the semantics of computer programs as the least fixed-points of some adequate higher order functions. The importance and elegance of such an approach for formal semantics is twohold:

(i) Associating a higher order function with each program is transparent and intuitively clear. Thus this method provides a mathematically clean and easy to understand formal semantics.

(ii) The mathematical analysis of such fixed points provides several very powerful induction methods for proving general properties about them.

The former claim can be seen in the recent success of the Mathematical Semantics of Programming Languages (13,14). Quite notable results have been obtained with respect to the latter, in the field of program verification (4,5,6,7,9). In this paper we will mainly focus on the latter.

The first (historically) important work in this direction was done by Park (4). He used the famous Tarski fixed-point theorem (1) of monotone functions from a complete lattice to itself as a mathematical foundation of this approach. Then he showed that, for each program, we can associate an adequate such monotone function whose least fixed-point is the semantics of that program. Also he presented a very general and powerful induction on the least fixed-point, the so-called fixed-point induction, and applied this induction for program verification to show that both the inductive assertion method for recursive programs and the recursion induction are special cases of fixed-point induction.

The only defect of Park's approach is that, using Tarski's fixed-point theorem, we can not feel the actual computation process involved in the program computation. In a sense, we can say that Tarski's fixed-point approach is too abstract for computer scientists. Scott overcame this weakness of the fixed-point approach. He observed that the semantics of computer programs should be defined

as the least fixed-points of continuous functions rather than monotone functions. He showed that for a continuous function from a complete lattice to itself, the least fixed point can be obtained effectively. Using the effectiveness of the least fixed-points, Scott discovered a very powerful induction method for proving general properties of the fixed-points. This method is called the Scott fixed-point induction method (5).

Various results followed after these two foundamental works. Manna and Morris discovered several other induction methods again using the effectiveness of Scott's least fixed-points (6,9). Manna also showed that Scott's fixed-point induction implies the inductive assertion methods for recursive programs and recursion induction, as in Park's work.

Like Manna and Morris, we will pursue Scott's fixed-point approach. The first main result in this paper involves logical semantics. The idea of logical semantics originates with Park (4). In his fixed-point approach, Park defined the semantics of recursive programs not mathematically but logically. In other words, for each recursive program, he associated not a monotone function but a monotone formula which extensionally characterizes a monotone function. Then he showed that the least fixed-point of such formula exists and that fixed-point extensionally defines the least fixed-point of the monotone function which the formula describes, But the work done so far in Scott's fixed-point approach takes the mathematical semantics approach (6,7,9). We observe that the fixed-point induction in the mathematical semantics approach enforces the logic to include the domain of functions as a part of individual domain (7), while this is not so in the logical semantics approach. Therefore the logical semantics makes the logic needed for carrying out the induction simpler than the mathematical semantics. For this reason, we will take the logical semantics approach with respect to Scott's fixed-point approach. In fact from section 2 to 4, we will show how to

associate a continuous formula for each program s.t. the logical formula intensionally defines the continuous function which is supposed to be associated with the program. We then observe that we can effectively obtain the least fixed-point of the continuous formula which intensionally defines the least fixed-point of the continuous function (12).

The second main result of this paper is a formulation of those induction methods of Scott, Manna, and Morris, in terms of the logical fixed-point theorem. We then apply these induction methods for program verification and observe that ad-hoc verification methods, like Floyd's inductive assertion method and King's symbolic execution method are special cases of these induction methods. An especially close investigation on the justification and advantage of King's method will be given. Note that unlike the work by Manna (6), we do not translate programs into recursive programs to carry out this sort of discussion. Therefore we belive our approach is more straightforward and natural.

The third main result is an interesting relation between the finiteness of logical semantics and the termination of program schemas. Actually the logical semantics of a program schema is a collection of first order infinitary formulas. A case study will leads us to a class of program schemas, the so-called terminating schemas, for which the logical semantics reduces a collection of first order finitary formulas (12).

The mathematical semantics approach can be formulated (in the logician's sense of formulation) as some sort of λ-calculus, like LCF (7). The logical semantic method is an alternative to LCF like application of Scott's fixed-point theorem in program verification. In fact LCF is so designed as to work even for higher order programming languages, in which a program treats programs as data. Obviously our programs in this paper are not higher ordered, and the logical semantics of higher order languages is not known yet. Thus our approach

sacrifices generality for simplicity and LCF scarifices simplicity for generality.

A more formal and detailed treatment of the discussion of section 2, 3, and 4 can be found in (12).

2. Scott's fixed-point theorem (5, 8)

Since we will take Scott's fixed-point approach, we need to know the mathematical foundation of this approach, which is the Scott's fixed-point theorem. In fact we will use a special version of his theory which is based not on complete lattices but on complete partial orderings.

A partially ordered set $(D,\leq)$ is a complete partial ordering (c.p.o.) iff D contains the minimal (w.r.t. $\leq$) element and every chain X in D has a least upper bound $\sqcup X$ in D. By chain X we mean a subset $\{x_i \mid i \in N\} \subseteq D$ s.t. $x_i \leq x_{i+1}$ for all i.

A function $f:D \rightarrow D$ is monotone iff $x_1 \leq x_2$ implies $f(x_1) \leq f(x_2)$ where D is a c.p.o. A monotone function $f:D \rightarrow D$ is continuous iff for every chain X in D, $\sqcup f(X) = f(\sqcup X)$.

Theorem Scott's fixed-point theorem

Any continuous function $f:D \rightarrow D$ has a least fixed point conv(f) given by the following: $conv(f) = \sqcup\{f^i(\perp) \mid i = 0,1,2,\ldots\}$ where $f^0(x) = \perp$ for any $x \in D$.

The formula for conv(f) suggests an induction rule for proving properties of conv(f). Presumably we can induce as follows:

"P(conv(f)) = T if $P(f^i(\perp)) = T$ for all i"

But, in general, the property P does not remain true in the limit. A good intuitive analogy to this matter can be found in the well known fact that no regular polygon is smooth but their limit, a circle, is smooth. But fortunately it has been known that a quite large interesting class of properties remain true in the limit. Actually, as we will see later, for proving programs partially correct, we can always use the above induction. We call properties which allow the above induction admissible (11).

For admissible properties, we actually have three different induction methods depending upon how we prove $P(f^i(\perp)) = T$ for all i (6,9).

(i) simple induction method

"P(conv(f)) = T if P($\perp$) = T and $(\forall i)[P(f^i(x)) \rightarrow P(f^{i+1}(x))] = T$"

(ii) truncation induction method

"P(conv(f)) = T if P($\perp$) = T and

$$(\forall i)\ [(\forall j \leq i)[P(f^j(x))] \rightarrow P(f^i(x))] = T"$$

(iii) Scott's fixed-point induction

"P(conv(f)) = T if P($\perp$) = T and $P(x) \rightarrow P(f(x)) = T$"

Note that the elegance of Scott's fixed-point indction is its simple inductive step.

3. The language

In this section we will set up the logical notions and notations needed later in this paper to define our logical semantics. Since our main purpose is to define computational object using mathematical logic, our language allows only a finite number of predicate and individual variables in formulas. Even though we are mainly interested in at most first order infinitary formulas to which we identify the semantics of programs, we need to establish a theory about these semantics. Therefore we need to extend our language to an at most second order language. Thus our language contains only at most second order infinitary formulas with finite variables. (Note: second order infinitary formulas will not be used in this paper.)

$\mathcal{F}(\mathcal{X})(\mathcal{x})$ where $\mathcal{X} = (X_1,\ldots,X_m)$ and $\mathcal{x} = (x_1,..,x_n)$ denotes that a formula $\mathcal{F}$ has at most m free predicate variables and n free individual variables.

An <u>interpretation</u> I of our language is a 4-tuple (D_I,F,P,C) where D_I is a set called the <u>domain</u> of individuals of I, F is a set of functions of D_I one for each basic function symbol, and P is a set of predicates of D_I one for each basic predicate symbol, and C is a subset of D_I one for each individual constant symbol of our language. Let S be a syntactic object of our language, then by S^I we mean the result of assigning the corresponding elements of I to each basic symbols of S.

Let $\mathcal{F}(\mathcal{X})(\mathcal{x})$ be a formula and $\mathcal{A}$ and $\mathcal{a}$ be vectors of informal predicates of D_I and of constants of D_I respectively. By $\mathcal{F}^I(\mathcal{A})(\mathcal{a})$ we mean the result of assigning $\mathcal{A}$ to $\mathcal{X}$ and $\mathcal{a}$ to $\mathcal{x}$ component-wise in $\mathcal{F}^I$. Note that we assume that the demension of $\mathcal{A}$ and $\mathcal{a}$ are the same as that of $\mathcal{X}$ and $\mathcal{x}$ respectively. By the <u>extension of $\mathcal{F}$ in</u> I, denoted by $\text{ext}^I(\mathcal{F})$, we mean the set of assignments:

$$\text{ext}^I(\mathcal{F}) = \{(\mathcal{A},\mathcal{a}) \mid \mathcal{F}^I(\mathcal{A})(\mathcal{a})\}.$$

Roughly speaking $\text{ext}^I(\mathcal{F})$ is a collection of informal predicates and individuals which makes $\mathcal{F}^I$ true.

Definition

Let $\mathfrak{F}(\mathfrak{X})(\mathfrak{x})$ be a formula. Then:

(i) $\mathfrak{F}$ is satisfied by an assignment $(\mathfrak{A}, \mathfrak{a})$ in I iff $(\mathfrak{A}, \mathfrak{a}) \in \text{ext}^I(\mathfrak{F})$.

(ii) $\mathfrak{F}$ is valid under I iff all possible assignments in I belong to $\text{ext}^I(\mathfrak{F})$.

(iii) $\mathfrak{F}$ is valid iff $\mathfrak{F}$ is valid under all admissible interpretations.

By admissible interpretations we mean interpretations which satisfy the equality axioms together with $T \neq F$.

We say that two formulas $\mathfrak{F}$ and $\mathfrak{G}$ are semantically equivalent under I, denoted by $\mathfrak{F} \equiv_I \mathfrak{G}$ iff $\text{ext}^I(\mathfrak{F}) = \text{ext}^I(\mathfrak{G})$. If $\mathfrak{F} \equiv_I \mathfrak{G}$ for all admissible I, then we say $\mathfrak{F}$ is semantically equal to $\mathfrak{G}$ and denote this by $\mathfrak{F} \equiv \mathfrak{G}$.

Lemma

(i) $\mathfrak{F} \equiv_I \mathfrak{G}$ iff $\mathfrak{F} \leftrightarrow \mathfrak{G}$ is valid under I.

(ii) $\mathfrak{F} \equiv \mathfrak{G}$ iff $\mathfrak{F} \leftrightarrow \mathfrak{G}$ is valid.

Because of this lemma we sometimes identify "$\equiv$" with "$\leftrightarrow$".

We have defined a sort of infromal (or semantic) substitution as assignment. Now we define formal (or syntactic) substitution. Let $\mathfrak{H}(X_1,\ldots,X_m)(x_1,\ldots,x_n)$ be a formula. Let $\mathfrak{G}_1,\ldots,\mathfrak{G}_m$ be formulas and let $\tau_1,\ldots,\tau_n$ be terms. By $\mathfrak{F}(\mathfrak{G}_1,\ldots,\mathfrak{G}_m)(\tau_1,\ldots,\tau_n)$, we mean the result of substituting $(\mathfrak{G}_1,\ldots,\mathfrak{G}_m)$ for $(X_1,\ldots,X_m)$ and $(\tau_1,\ldots,\tau_n)$ for $(x_1,\ldots,x_n)$. The above substitution should be done in such a way that no free variable in $\mathfrak{G}_1,\ldots,\mathfrak{G}_m,\tau_1,\ldots,\tau_n$ will be bound in $\mathfrak{H}(\mathfrak{G}_1,\ldots,\mathfrak{G}_m)(\tau_1,\ldots,\tau_n)$.

Definition

Let $\zeta(X_1,\ldots,X_m)(\mathfrak{x})$ be a formula. Let $\mathfrak{G}_1,\ldots,\mathfrak{G}_m$ be formulas.

(i) $\mathfrak{G}_1,\ldots,\mathfrak{G}_m$ satisfy ζ under I iff $\zeta(\mathfrak{G}_1,\ldots,\mathfrak{G}_m)(\mathfrak{x})$ is valid under I.

(ii) $\mathfrak{G}_1,\ldots,\mathfrak{G}_m$ satisfy ζ iff $\zeta(\mathfrak{G}_1,\ldots,\mathfrak{G}_m)(\mathfrak{x})$ is valid.

Let $\zeta_i(X_1,\ldots,X_N)(\mathfrak{x})$ be finitary formulas $(1 \leq i \leq N)$. Then we call the following set of formulas, a system of formula equations.

$$\zeta_1(X_1,\ldots,X_N)(\alpha) \equiv X_1(\alpha)$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$\zeta_N(X_1,\ldots,X_N)(\alpha) \equiv X_N(\alpha)$$

A solution to this system is a set of formulas $\xi_1,\ldots,\xi_N$ satisfying these equations. A solution under I of this system is a set of formulas $\xi_1,\ldots,\xi_N$ satisfying these equations under I.

Later in this paper we will associate a system of formula equations with a program schema and identify the semantics of this program schema to the least solution of this system of formula equations. In other words, we associate $(\zeta_1,\ldots,\zeta_N)$ with a program schema and define the semantics of the schema as the least fixed point of $(\zeta_1,\ldots,\zeta_N)$.

4. A least solution of a system of formula equations

In this section we will define a class of systems of formula equations for which the effective least solutions exist. The class which will be discussed here is that of continuous systems. A discussion which is essentially equivalent but based on the monotone systems can be found in (4).

To introduce the notion of continuity for formulas, and hence apply Scott's fixed-point theorem to obtain the least fixed-point of them, we somehow have to interpret a formula as a function, since the fixed-point theorem is described in terms of functions. Suppose $\mathcal{L}(X)(\mathfrak{A})$ is a formula. We can regard this formula as a statement, under an admissible interpretation I, of a function $C_I^{\mathcal{L}}$ of relations such that: $C_I^{\mathcal{L}}(A) = \{\mathfrak{a} \mid \mathcal{L}^I(A)(\mathfrak{a})\}$.

Let us define $\mathcal{L}(X)(\mathfrak{A})$ to be <u>continuous</u> iff $C_I^{\mathcal{L}}$ is continuous for all admissible I. If X is a (demension of $\mathfrak{A}$)-ary predicate variable then $C_I^{\mathcal{L}}$ has the least fixed point $\mathrm{conv}(C_I^{\mathcal{L}})$, whenever $\mathcal{L}$ is continuous. Using the equation:

$$\mathrm{conv}(C_I^{\mathcal{L}}) = \bigsqcup \{(C_I^{\mathcal{L}})^i(\bot) \mid i = 0,1,2,\ldots\}$$

we can easily prove:

$$\mathrm{conv}(C_I^{\mathcal{L}}) = \mathrm{ext}^I(\mathrm{Conv}(\mathcal{L})(\mathfrak{A}))$$

where $\mathrm{Conv}(\mathcal{L})(\mathfrak{A}) = F \vee \mathcal{L}(F)(\mathfrak{A}) \vee \mathcal{L}^2(F)(\mathfrak{A}) \vee \ldots\ldots$

where $\mathcal{L}^n(F)(\mathfrak{A}) = \mathcal{L}(\mathcal{L}^{n-1}(F)(\mathfrak{A}))(\mathfrak{A})$. Note that the above relation states that $\mathrm{Conv}(\mathcal{L})(\mathfrak{A})$ is a logical description of $\mathrm{conv}(C_I^{\mathcal{L}})$.

Applying Scott's fixed-point thorem to $C_I^{\mathcal{L}}$, we can easily prove the following theorem:

<u>Theorem</u> <u>Scott's fixed-point theorem for formulas</u>

Let $\mathcal{L}(X)(\mathfrak{B})$ be continuous and let X be (demension of $\mathfrak{B}$)-ary. Then $\mathrm{Conv}(\mathcal{L})(\mathfrak{B})$ is the extensionally least solution of the following system of equations:

$$\mathcal{L}(X)(\mathfrak{B}) \equiv X(\mathfrak{B}).$$

We can easily extend the above results to the case of simultaneous formula equations. We first associate the following function:

$C_I^{\mathcal{L}}(\mathbb{A}) = (C_I^{\mathcal{L}_1}(\mathbb{A}),\ldots,C_I^{\mathcal{L}_N}(\mathbb{A}))$ where $C_I^{\mathcal{L}_i}(\mathbb{A}) = \{a \mid \mathcal{L}_i^I(\mathbb{A})(a)\}$,

with $\mathcal{L} = (\mathcal{L}_1,\ldots,\mathcal{L}_N)$ where $\mathcal{L}_i = \mathcal{L}_i(X_1,\ldots,X_N)(\mathcal{X})$ and the X_j are (demension of $\mathcal{X}$)-ary. $(1 \le i,j \le N)$. We define $\mathcal{L}$ to be continuous iff $C_I^{\mathcal{L}}$ is continuous for all interpretations I.

Lemma

Suppose $\mathcal{L}$ is continuous, then:

$$\mathrm{conv}(C_I^{\mathcal{L}}) = (\mathrm{conv}_1(C_I^{\mathcal{L}}),\ldots,\mathrm{conv}_N(C_I^{\mathcal{L}}))$$

where $\mathrm{conv}_i(C_I^{\mathcal{L}}) = \bigsqcup\{((C_I)^k(\bot))_i \mid k = 0,1,2,\ldots\}$ and $(Y)_i$ denotes the ith component of Y.

This lemma is very important because it assures us that to obtain each component of the limit, we can take the limit component-wise.

Definition

Let $\mathcal{L} = (\mathcal{L}_1,\ldots,\mathcal{L}_N)$ where $\mathcal{L}_i(X_1,\ldots,X_N)(\mathcal{X})$ and X_j are (demension of $\mathcal{X}$)-ary $(1 \le i,j \le N)$. Define $\mathrm{Conv}_i(\mathcal{L})(\mathcal{X})$ by:

$$\mathrm{Conv}_i(\mathcal{L})(\mathcal{X}) = F \vee \mathcal{L}_i^1(\mathbb{F})(\mathcal{X}) \vee \mathcal{L}_i^2(\mathbb{F})(\mathcal{X}) \vee \ldots \quad \text{with } \mathbb{F} = (\overbrace{F,\ldots,F}^{N}),$$

where $\mathcal{L}_i^1(\mathbb{F})(\mathcal{X}) = \mathcal{L}_i(\mathbb{F})(\mathcal{X})$

$$\mathcal{L}_i^n(\mathbb{F})(\mathcal{X}) = \mathcal{L}_i(\mathcal{L}^{n-1}(\mathbb{F})(\mathcal{X}))(\mathcal{X})$$

and $\mathcal{L}^k(\mathbb{F})(\mathcal{X}) = (\mathcal{L}_1^k(\mathbb{F})(\mathcal{X}),\ldots,\mathcal{L}_N^k(\mathbb{F})(\mathcal{X}))$.

Let $\mathrm{Conv}(\mathcal{L})(\mathcal{X}) = (\mathrm{Conv}_1(\mathcal{L})(\mathcal{X}),\ldots,\mathrm{Conv}_N(\mathcal{L})(\mathcal{X}))$.

Now we can prove $\mathrm{conv}_i(C_I^{\mathcal{L}}) = \mathrm{ext}^I(\mathrm{Conv}_i(\mathcal{L})(\mathcal{X}))$ if $\mathcal{L}$ is continuous. Therefore, intuitively, we can think that the formula $\mathrm{Conv}_i(\mathcal{L})(\mathcal{X})$ is a logical description of the function $\mathrm{conv}_i(C_I^{\mathcal{L}})$. Hence using Scott's fixed-point theorem for $\mathrm{conv}_i(C_I^{\mathcal{L}})$, we obtain the following theorem:

Theorem Scott's fixed-point theorem for formulas (extended)

Let $\mathcal{L} = (\mathcal{L}_1,\ldots,\mathcal{L}_N)$ be continuous where $\mathcal{L}_i = \mathcal{L}_i(X_1,\ldots,X_N)(\mathcal{X})$ and X_j is (demension of $\mathcal{X}$)-ary $(1 \le i,j \le N)$. Then $\mathrm{Conv}(\mathcal{L})(\mathcal{X})$ is the extensionally least solution of the following system of formula equations:

$$\mathcal{L}_1(\mathcal{X})(\mathcal{Z}) \equiv X_1(\mathcal{Z})$$

.....

$$\mathcal{L}_N(\mathcal{X})(\mathcal{Z}) \equiv X_N(\mathcal{Z}).$$

In other words, it is the least fixed-point of $\mathcal{L}$.

The definition of $\mathrm{Conv}(\mathcal{L})(\mathcal{Z})$ suggests the following induction methods for proving properties of $\mathrm{Conv}(\mathcal{L})(\mathcal{Z})$.

A formula is said to be admissible under I iff the following formula is valid under I: $(\forall i)[\mathcal{F}(\mathcal{L}^i(\mathcal{F})(\mathcal{Z}))(\mathcal{Z})] \rightarrow \mathrm{Conv}(\mathcal{L})(\mathcal{Z})$.

$\mathcal{F}$ is admissible iff the above formula is valid. In the following inductions we assume that $\mathcal{F}$ is admissible (under I).

The simple induction method

$\mathcal{F}(\mathrm{Conv}(\mathcal{L})(\mathcal{Z}))(\mathcal{Z})$ is valid (under I) if $\mathcal{F}(\mathcal{F})(\mathcal{Z})$ is valid (under I) and $(\forall i)[\mathcal{F}(\mathcal{L}^i(\mathcal{X})(\mathcal{Z}))(\mathcal{Z}) \rightarrow \mathcal{F}(\mathcal{L}^{i+1}(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})]$ is valid (under I).

The truncation induction method

$\mathcal{F}(\mathrm{conv}(\mathcal{L})(\mathcal{Z}))(\mathcal{Z})$ is valid (under I) if $\mathcal{F}(\mathcal{F})(\mathcal{Z})$ is valid (under I) and $(\forall i)[(\forall j \leq i)[\mathcal{F}(\mathcal{L}^j(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})] \rightarrow \mathcal{F}(\mathcal{L}^i(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})]$ is valid (under I).

The Scott's fixed-point induction method

$\mathcal{F}(\mathrm{Conv}(\mathcal{L})(\mathcal{Z}))(\mathcal{Z})$ is valid (under I) if $\mathcal{F}(\mathcal{F})(\mathcal{Z})$ is valid (under I) and the validity of $\mathcal{F}(\mathcal{X})(\mathcal{Z})$ implies that of $\mathcal{F}(\mathcal{L}(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})$ (under I).

When we think of the relation: $\mathrm{Conv}_i(\mathcal{L})(\mathcal{Z}) = \bigvee_{k=0}^{\infty} \mathcal{L}_i^k(\mathcal{F})(\mathcal{Z})$, we can modify the truncation induction as follows:

Modified truncation induction

$\mathcal{F}(\mathrm{Conv}_i(\mathcal{L})(\mathcal{Z}))(\mathcal{Z})$ is valid (under I) if $\mathcal{F}(F)(\mathcal{Z})$ is valid (under I) and $(\forall k)[(\forall j \leq k)[\mathcal{F}(\mathcal{L}_i^j(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})] \rightarrow \mathcal{F}(\mathcal{L}_i^k(\mathcal{X})(\mathcal{Z}))(\mathcal{Z})]$ is valid (under I).

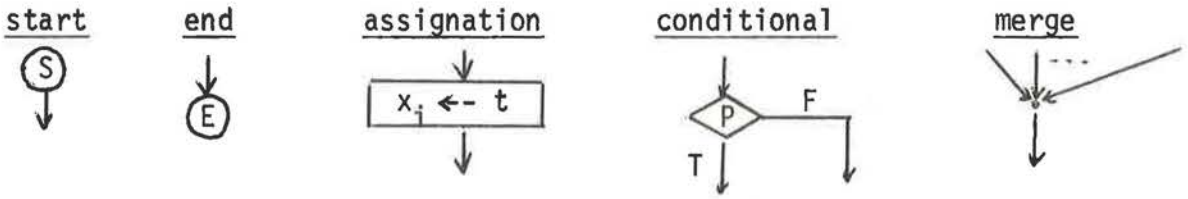
Note that in the above inductions, the logic needed is just ordinary first order logic. Especially note that we do not need to treat functions as individuals as in section 2, just because we express functions by formulas.

5. Fixed-point semantics of program schemas

We will define a class of syntactic objects, called the program schema. We first associate a system of function equation schema to each program schema and define the semantics of the program schema as the least solution of that system of function equation schemas (3). Hence we obtain a mathematical semantics of the program schema. Next we associate with each program schema a system of formula equations in such a way that the least solution of this system coincides with the mathematical semantics of this program schema. Hence we obtain a logical semantics which is consistent with the mathematical semantics in Floyd's sense (2,12).

5.1 Program schemas

Program schemas are composed of the following program elements:



where x_i is an individual variable, t is a term, and P is a formula without predicate variables.

A program schema is a collection of program elements satisfying:

(i) it contains exactly one start and one end.

(ii) each edge originates from one program element and proceeds into one another program element.

A free individual variable in a program schema is called a program variable. To denote that a syntactic object S of a program schema has at most n free program variables $\mathcal{X} = (x_1,\ldots,x_n)$, we write $S(\mathcal{X})$.

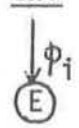
The interpretations of program schemas are the same as those of our language in section 3. To denote a syntactic object S of a program schema interpreted under I, we write S^I.

5.2 Mathematical semantics

When we execute an interpreted program schema $\mathcal{P}^I(\mathcal{X})$, the program variables $\mathcal{X}$ are to be bound to elements of D_I, at each program point (edge). We call a vector of elements of D_I bound to $\mathcal{X}$ a state-vector of $\mathcal{P}^I(\mathcal{X})$ (3). Essentially the semantics of an interpreted program schema is defined as a function of state-vectors. In usual schemata theory this function is defined operationally in terms of control flow (10). But in this paper we will define it mathematically.

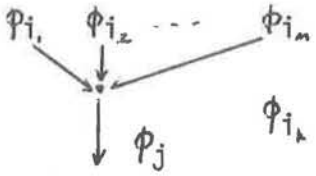
Given a program schema $\mathcal{P}(\mathcal{X})$, let us associate a function variable with edge (program point) of $\mathcal{P}(\mathcal{X})$. For each program element of $\mathcal{P}(\mathcal{X})$, let us associate the following function equation schema:

end



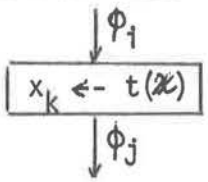
$\phi_i(\mathcal{X}) = \mathcal{X}$

merge



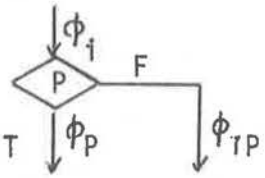
$\phi_{i_k} = \phi_j \quad (1 \leq k \leq m)$

assignation



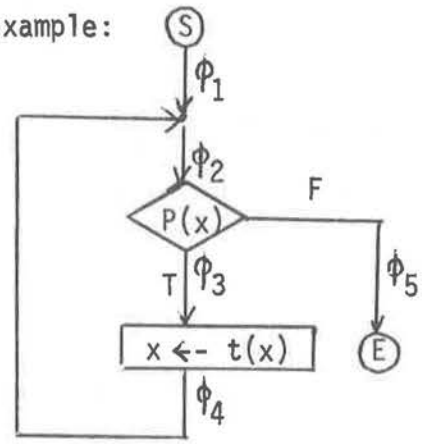
$\phi_i(\mathcal{X}) = \phi_j(x_1,\ldots,x_{k-1},t(\mathcal{X}),x_{k+1},\ldots,x_n)$

conditional



$\phi_i(\mathcal{X}) = \underline{\text{if}}\ P(\mathcal{X})\ \underline{\text{then}}\ \phi_P(\mathcal{X})\ \underline{\text{else}}\ \phi_{\neg P}(\mathcal{X})$

Thus for $\mathcal{P}(\mathcal{X})$, we have a system $\mathcal{F}_{\mathcal{P}}(\Phi)(\mathcal{X})$ of function equation scheams. For example:



$\phi_1(x) = \phi_2(x)$

$\phi_2(x) = \underline{\text{if}}\ P(x)\ \underline{\text{then}}\ \phi_3(x)\ \underline{\text{else}}\ \phi_5(x)$

$\phi_3(x) = \phi_4(t(x))$

$\phi_4(x) = \phi_2(x)$

$\phi_5(x) = x$

Let $(\phi_1^I,\ldots,\phi_m^I)$ be the least solution of $\Phi_{\mathcal{P}}^I(\varphi_1,\ldots,\varphi_m)(\mathcal{X})$, where $\Phi_{\mathcal{P}}^I(\varphi_1,\ldots,\varphi_m)(\mathcal{X})$ is the interpretation under I of $\Phi_{\mathcal{P}}(\varphi_1,\ldots,\varphi_m)(\mathcal{X})$. Then it is easy to observe that ϕ_i^I is the function computed by $\mathcal{P}^I(\mathcal{X})$ of state-vectors from the ith edge of $\mathcal{P}^I(\mathcal{X})$ to the end of $\mathcal{P}^I(\mathcal{X})$ $(1\leq i\leq m)$ (3). This is the so called tail function. Thus we can safely regard $(\Phi_1^I,\ldots,\Phi_m^I)$, the leastsolution of $\Phi_{\mathcal{P}}^I(\Phi)(\mathcal{X})$ as the mathematical semantics of $\mathcal{P}^I(\mathcal{X})$. By the least solution of $\Phi_{\mathcal{P}}(\Phi)(\mathcal{X})$, we mean the set $\{(\phi_1^I,\ldots,\phi_m^I) \mid I \text{ is admissible}\}$. We define the mathematical semantics of $\mathcal{P}(\mathcal{X})$ by the least solution of $\Phi_{\mathcal{P}}(\Phi)(\mathcal{X})$.

5.3 Some properties of program schemas

Let $\mathcal{P}(\mathcal{X})$ be a program schema and ϕ_1^I be the tail function associated with the start edge of $\mathcal{P}^I(\mathcal{X})$. Then we say $\mathcal{P}(\mathcal{X})$ terminates under I (or $\mathcal{P}^I(\mathcal{X})$ terminates) iff ϕ_1^I is total. We say $\mathcal{P}(\mathcal{X})$ terminates iff it terminates under all admissible interpretations.

Let $\mathcal{P}_1(\mathcal{X})$ and $\mathcal{P}_2(\mathcal{X})$ be program schemas, and $\phi_{1_1}^I$, $\phi_{1_2}^I$ be the tail functions for the start edges of $\mathcal{P}_1^I(\mathcal{X})$ and $\mathcal{P}_2^I(\mathcal{X})$ respectively. Then we say that $\mathcal{P}_1(\mathcal{X})$ and $\mathcal{P}_2(\mathcal{X})$ are equivalent under I (or $\mathcal{P}_1^I(\mathcal{X})$ and $\mathcal{P}_2^I(\mathcal{X})$ are equivalent) iff $\phi_{1_1}^I$ and $\phi_{1_2}^I$ are equal as functions. We say $\mathcal{P}_1(\mathcal{X})$ and $\mathcal{P}_2(\mathcal{X})$ are equivalent iff they are so under all admissible interpretations.

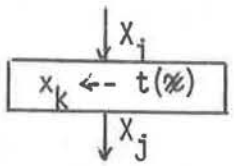
5.4 The logical semantics

Here we will associate a system of formula equations to each program schema and observe that the least solution of this system of equations coincides with the mathematical semantics of the program schema. Unlike Park's logical semantics for recursive schema, the least solution here does not describe the mathematical semantics directly as its extention. It will be observed that the logical semantics describes the mathematical semantics as a set of assertions which is consistent with the mathematical semantics (or the program schema itself). The notion of consistent assertions can be found in (2).

First, given a program schema $\mathcal{P}(\mathcal{X})$, we associate a predicate variable to

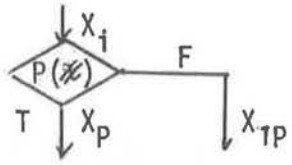
each edge of $\mathcal{P}(\mathfrak{A})$. Then for each program element, we associate the following first order continuous formula:

assignation



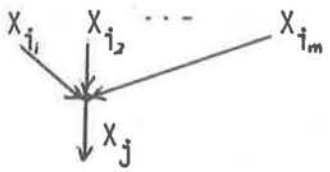
$$X_i(\mathfrak{A}) \equiv (\exists y)[x_k = t(x_1,\ldots,x_{k-1},y,x_{k+1},\ldots,x_n) \;\&\; X_i(x_1,\ldots,x_{k-1},y,x_{k+1},\ldots,x_n)]$$

conditional



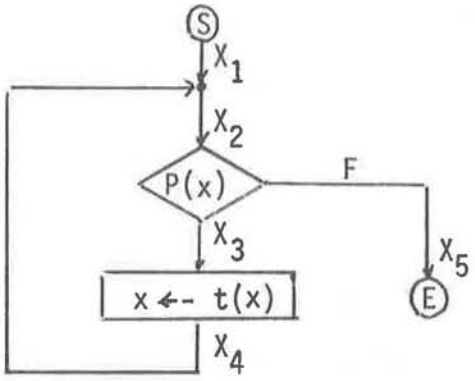
$$X_P(\mathfrak{A}) \equiv X_i(\mathfrak{A}) \;\&\; P(\mathfrak{A}) \qquad X_{\neg P}(\mathfrak{A}) \equiv X_i(\mathfrak{A}) \;\&\; \neg P(\mathfrak{A})$$

merging



$$X_j(\mathfrak{A}) \equiv X_{i_1}(\mathfrak{A}) \vee X_{i_2}(\mathfrak{A}) \vee \ldots \vee X_{i_m}(\mathfrak{A})$$

Therefore for each program schema $\mathcal{P}(\mathfrak{A})$, we have a system $\Psi_{\mathcal{P}}(X)(\mathfrak{A})$ of formula equations. For example:



$$X_2(x) \equiv X_1(x) \vee X_4(x)$$
$$X_3(x) \equiv X_2(x) \;\&\; P(x)$$
$$X_4(x) \equiv (\exists y)[x = t(y) \;\&\; X_3(y)]$$
$$X_5(x) \equiv X_2(x) \;\&\; \neg P(x)$$

As it can be seen in the above example, we obtain n equations for n+1 predicate variables. Thus we have to fix one of them. Usually we fix X_1 to some predicate which describes the domain of the program.

Let $\mathcal{P}(\mathfrak{A})$ be a program schema and $\Psi_{\mathcal{P}}(X_1,\ldots,X_N)(\mathfrak{A})$ be its system of formula

equations. Let $(Q_1,\ldots,Q_N)$ be the least solution of $\Phi_{\mathcal{P}}(X_1,\ldots,X_N)(\mathcal{X})$. Then $ext^I(Q_i)$ can be interpreted as the set of state-vectors to which the program variables $\mathcal{X}$ can be bound at the ith edge when we execute the program $\mathcal{P}^I(\mathcal{X})$ on the initial program states $ext^I(Q_1)$. Therefore we can regard $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ as a description of the computation of $\mathcal{P}^I(\mathcal{X})$ not on each initial program state but on the set of initial program states (12). In this sense we can say that the least solution of $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ describes the semantics of $\mathcal{P}(\mathcal{X})$. The following formal discussion will justify this intuitive argument.

Theorem

Let $\mathcal{P}(\mathcal{X})$ be a program schema and $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ be the system of formula equations associated with $\mathcal{P}(\mathcal{X})$. Let the ith and jth edge of $\mathcal{P}(\mathcal{X})$ be the input and output edges of a same program element in $\mathcal{P}(\mathcal{X})$ respectively. Then:

$$\phi_i^I(ext^I(Q_i)) \subseteq \phi_j^I(ext^I(Q_j)) \quad \text{for all admissible I,}$$

where ϕ_i^I and ϕ_j^I are the tail functions for the ith and jth edges of $\mathcal{P}^I(\mathcal{X})$ respectively, and Q_i and Q_j are the least solutions of $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ for the ith and jth edges respectively.

Note that the above theorem states that the least solution of $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$, i.e. the logical semantics of $\mathcal{P}(\mathcal{X})$, is consistent with $\mathcal{P}(\mathcal{X})$ in Floyd's sense (2,12). Also observing the structure of the program scheam's syntax, we obtain the following theorem.

Theorem

Let $\mathcal{P}(\mathcal{X})$ and $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ be as above, and let Q_1 and Q_N be the least solution of $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ for the start and end edges of $\mathcal{P}(\mathcal{X})$ respectively. Also let ϕ_1^I and ϕ_N^I be the tail functions for each of these edges respectively. Then:

$$\phi_1^I(ext^I(Q_1)) = \phi_N^I(ext^I(Q_N)) = ext^I(Q_N)$$

The next theorem tells us that the logical semantics gives the same semantics for equivalent program schemas.

Theorem

Let $\mathcal{P}_1(\mathcal{K})$ and $\mathcal{P}_2(\mathcal{K})$ be equivalent under I. Let $(Q_{1_1},\ldots,Q_{N_1})$ and $(Q_{1_2},\ldots,Q_{N_2})$ be the least solutions for $\mathfrak{H}_{\mathcal{P}_1}$ and $\mathfrak{H}_{\mathcal{P}_2}$ respectively. Then:

$$Q_{N_1} \equiv_I Q_{N_2} \quad \text{if} \quad Q_{1_1} \equiv_I Q_{1_2} .$$

6. Finite semantics and termination of program schemas.

In the previous section, we have observed that the logical semantics of a program schema $\mathcal{P}(\mathcal{X})$ is the least solution of the corresponding system of formula equations $\Phi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$. As we have seen in section 4, therefore the semantics of $\mathcal{P}(\mathcal{X})$ is a set of first order infinitary formulas. In this section, we will observe those cases in which these infinitary formulas reduce to first order finitary formulas. The following lemma is the mathematical foundation of this sort of reduction.

Lemma

Let $f:D \to D$ be continuous. Suppose, for some m, $f^m(\perp) = f^{m+1}(\perp)$. Then:

$$conv(f) = f^m(\perp).$$

Applying the above lemma to $C_I^{\mathcal{L}}$, we have the following theorem:

Theorem

Let $\mathcal{L} = (\mathcal{L}_1,\ldots,\mathcal{L}_N)$ where $\mathcal{L}_i = \mathcal{L}_i(X_1,\ldots,X_N)(\mathcal{X})$ and X_j is (demension of $\mathcal{X}$)-ary $(1 \le i,j \le N)$. Suppose $\mathcal{L}$ is continuous. Also suppose that for some m, $\mathcal{L}_i^m(\mathbb{F})(\mathcal{X}) \equiv_I \mathcal{L}_i^{m+1}(\mathbb{F})(\mathcal{X})$. Then:

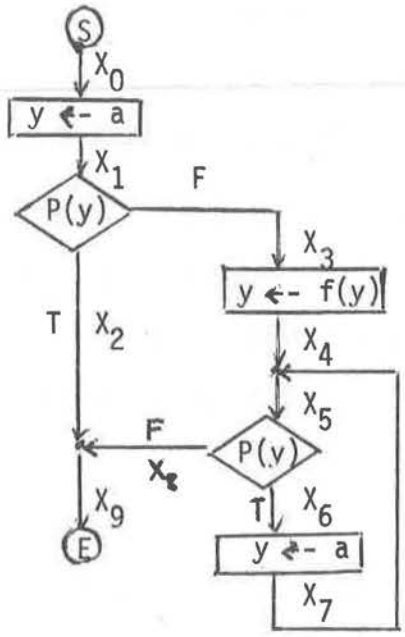
$$Conv_i(\mathcal{L})(\mathcal{X}) \equiv_I \mathcal{L}_i^m(\mathbb{F})(\mathcal{X}).$$

The above theorem gives us a sufficient condition for the finiteness of $Conv_i(\mathcal{L})(\mathcal{X})$. Thus our next question is what is a class of program schemas for which the logical semantics is finite. Actually we have a very important class of program schemas whose logical semantics is finite.

Theorem

If a program schema terminates then the logical semantics of it is finite, i.e. the logical semantics of it reduces to a set of first order finitary formulas.

For example, the following program schema terminates:



$X_1(y) \equiv (\exists z)[y = a \;\&\; Q_0(z)] \equiv \zeta_1(\mathcal{X})(y)$

$X_2(y) \equiv X_1(y) \;\&\; P(y) \equiv \zeta_2(\mathcal{X})(y)$

$X_3(y) \equiv X_1(y) \;\&\; \neg P(y) \equiv \zeta_3(\mathcal{X})(y)$

$X_4(y) \equiv (\exists z)[y = f(z) \;\&\; X_3(z)] \equiv \zeta_4(\mathcal{X})(y)$

$X_5(y) \equiv X_4(y) \vee X_7(y) \equiv \zeta_5(\mathcal{X})(y)$

$X_6(y) \equiv X_5(y) \;\&\; P(y) \equiv \zeta_6(\mathcal{X})(y)$

$X_7(y) \equiv (\exists z)[y = a \;\&\; X_6(z)] \equiv \zeta_7(\mathcal{X})(y)$

$X_8(y) \equiv X_5(y) \;\&\; \neg P(y) \equiv \zeta_8(\mathcal{X})(y)$

$X_9(y) \equiv X_8(y) \vee X_2(y) \equiv \zeta_9(\mathcal{X})(y)$

We can easily check that $\zeta_i^{10}(\mathcal{F})(y) = \zeta_i^{9}(\mathcal{F})(y)$ $(1 \leq i \leq 9)$. Therefore we have:

$$\mathrm{Conv}_i(\zeta)(y) \equiv \zeta_i^{9}(\mathcal{F})(y) \quad (1 \leq i \leq 9).$$

One might expect that if an interpreted program schema $\mathcal{P}^I(\mathcal{X})$ terminates, then the logical semantics of $\mathcal{P}^I(\mathcal{X})$ is finite. But this is not true in general. More precisely, let $\Xi_{\mathcal{P}}(\mathcal{X})(\mathcal{X})$ be as follows:

$$\zeta_1(\mathcal{X})(\mathcal{X}) \equiv X_1(\mathcal{X})$$

.....

$$\zeta_N(\mathcal{X})(\mathcal{X}) \equiv X_N(\mathcal{X}).$$

Then even if $\mathcal{P}^I(\mathcal{X})$ terminates, in general, "for some k, $\zeta_j^{k}(\mathcal{F})(\mathcal{X}) \equiv_I \zeta_j^{k+1}(\mathcal{F})(\mathcal{X})$ $(1 \leq j \leq N)$" is not true. The reason is that, for such $\mathcal{P}^I(\mathcal{X})$, the least upper bound of the computation steps for each initial value of $\mathcal{X}$ might be infinite. Note this least upper bound for a terminating $\mathcal{P}(\mathcal{X})$ is always finite and is the steps of the symbolic execution of $\mathcal{P}(\mathcal{X})$.

7. Program verification.

With the logical semantics in hand, we are now in a position to prove properties of program schemas. In this paper we will treat only a class of properties, the so called partial correctness. Informally we can describe these properties as follows:

"A program schema $\mathcal{P}(\mathcal{X})$ is partially correct with respect to a pair of formulas (Φ, Ψ) under I iff when we start the computation of $\mathcal{P}^I(\mathcal{X})$ in the initial state $\mathcal{X}$ for which $\Phi^I(\mathcal{X})$ is true, then if the computation terminates in a state $\mathcal{X}'$ then $\Psi^I(\mathcal{X}')$ is also true." (2,6)

Using the logical semantics, we can formally define partial correctness.

Definition

Let $\mathcal{P}(\mathcal{X})$ be a program schema and $(Q_1,\ldots,Q_N)$ be a least solution of the corresponding system $\mathcal{F}_{\mathcal{P}}(X)(\mathcal{X})$. Then $\mathcal{P}(\mathcal{X})$ is partially correct w.r.t (Φ, Ψ) under I iff :

$$\Phi(\mathcal{X}) \equiv Q_1(\mathcal{X}) \;\&\; (Q_N(\mathcal{X}) \rightarrow \Psi(\mathcal{X}))$$

is valid under I, where the 1st and Nth edges are the start and the end edges respectively.

If the logical semantics of $\mathcal{P}(\mathcal{X})$ is fineite, then proving $\mathcal{P}^I(\mathcal{X})$ partially correct is simply theorem proving in first order logic. But if it is not finite, then, in general, we have to prove the fact in first order infinitary logic (16). Actually the infinitary logic needed for this purpose is very restricted and it might be worthwhile to investigate this sort of logic. But we will not take this approach in this paper. Actually the induction methods discussed in section 4 provide finitary proof methods and they work well practically. In fact we will observe that several proof methods proposed for program verification from practical view point, like the inductive assertion method of Floyd (2), the symbolic execution method of King (15), turn out to be special cases of these induction methods.

Obviously, from the previous definition, to prove $\mathcal{P}(\mathcal{X})$ partially correct w.r.t. (φ, Ψ) under I, we need to do the following two steps:

<u>step 1</u> : Solve the corresponding system:

$$\zeta_1(\mathcal{X})(\mathcal{X}) \equiv X_1(\mathcal{X})$$

.....

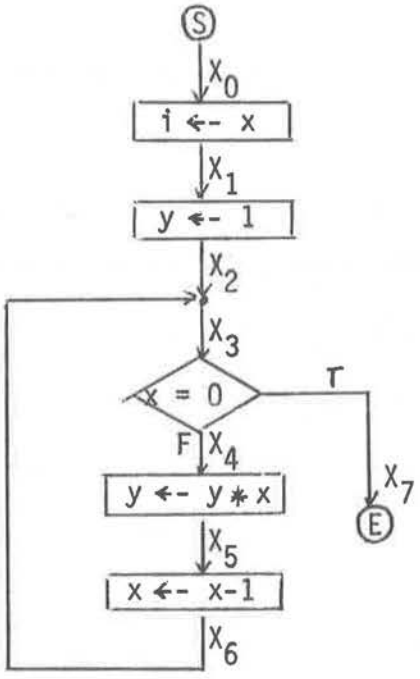
$$\zeta_N(\mathcal{X})(\mathcal{X}) \equiv X_N(\mathcal{X})$$

with $X_1(\mathcal{X}) \equiv \varphi(\mathcal{X})$.

<u>step 2</u> : Prove $\mathrm{Conv}_N(\zeta_1)(\mathcal{X}) \rightarrow \Psi(\mathcal{X})$ is valid under I.

Note the formula $\mathcal{F}(\mathcal{X})(\mathcal{X}) \equiv \bigwedge_{k=1}^{N} (X_k(\mathcal{X}) \rightarrow P_k(\mathcal{X}))$ is admissible (11). Therefore to prove the partial correctness, we can use the induction methods in section 4. Also note, for this $\mathcal{F}$, $\mathcal{F}(\mathbb{F})(\mathcal{X})$ is vacuously valid under I. We will see how these inductions work through several examples.

(<u>example 1</u>) The following program schema computes the factorial function, under the usual interpretation I_N of natural numbers. We will prove this as follows:



$$X_1(x,y,i) \equiv i = x \;\&\; x \geq 0 \equiv \zeta_1$$

$$X_2(x,y,i) \equiv (\exists z)[y = 1 \;\&\; X_1(x,z,i)] \equiv \zeta_2$$

$$X_3(x,y,i) \equiv X_2(x,y,i) \vee X_6(x,y,i) \equiv \zeta_3$$

$$X_4(x,y,i) \equiv X_3(x,y,i) \;\&\; x \neq 0 \equiv \zeta_4$$

$$X_5(x,y,i) \equiv (\exists z)[y = z * x \;\&\; X_4(x,z,i)] \equiv \zeta_5$$

$$X_6(x,y,i) \equiv (\exists z)[x = z-1 \;\&\; X_5(z,y,i)] \equiv \zeta_6$$

$$X_7(x,y,i) \equiv X_3(x,y,i) \;\&\; x = 0 \equiv \zeta_7$$

We will apply Scott's fixed-point induction rule.

We first assume the followings are valid under I_N.

$X_1(x.y.i) \rightarrow (i = x \;\&\; x \geq 0)$

$X_2(x,y,i) \rightarrow (y = 1 \;\&\; x = i \;\&\; x \geq 0)$

$X_3(x,y,i) \rightarrow (x = 0 \rightarrow y = i!) \;\&\; (x > 0 \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; x < i))$

$X_4(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; 0 < x < i)$

$X_5(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) * x \;\&\; 0 < x < i)$

$X_6(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; 0 \leq x < i-1)$

Using these induction assumptions we can easily prove the followings are valid under I_N:

$\zeta_1(\mathcal{X})(x,y,i) \rightarrow (i = x \;\&\; x \geq 0)$

$\zeta_2(\mathcal{X})(x,y,i) \rightarrow (y = 1 \;\&\; i = x \;\&\; x \geq 0)$

$\zeta_3(\mathcal{X})(x,y,i) \rightarrow ((x = 0 \rightarrow y = i!) \;\&\; (x > 0 \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; x < i)))$

$\zeta_4(\mathcal{X})(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; 0 < x < i)$

$\zeta_5(\mathcal{X})(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) * x \;\&\; 0 < x < i)$

$\zeta_6(\mathcal{X})(x,y,i) \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; 0 \leq x < i-1)$

Therefore by Scott's fixed-point induction the following formula is valid under I_N: $\mathrm{Conv}_3(\zeta)(x,y,i) \rightarrow ((x = 0 \rightarrow y = i!) \;\&\; (x > 0 \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; x < i)))$.

Therefore $\mathrm{Conv}_7(\zeta)(x,y,i) \rightarrow y = i!$ is valid under I_N. Therefore we have proved that the program schema is partially correct w.r.t. $(x \geq 0, y = i!)$ under I_N. Note that this proof is an elegant version of Floyd's inductive assertion method (2).

(example 2) For the same program schema and the same interpretation I_N as in the previous example, we prove the same thing using Morris's truncation induction method.

By manupulation of the system of formula equations, we have:

$X_3(x,y,i) \equiv (y = 1 \;\&\; x = i \;\&\; x \geq 0) \;\vee$

$$(\exists w)[y = w * (x+1) \;\&\; X_3(x+1, w, i) \;\&\; x+1 \neq 0] \;\ldots\; (i).$$

Note that this manupulation corresponds to ascending 3 steps on the chain for $Conv_3(\mathcal{L}_1)(\mathcal{K})$.

We will proceed with the induction by assuming the induction assumption on the right side X_3, and prove the assumption on the left side X_3. Note, since (i) corresponds to 3 steps ascending on the chain for $Conv_3(\mathcal{L}_1)(\mathcal{K})$, the induction hypothesis is not that of simple or Scott's fixed-point inductions, but that of truncation induction. Therefore the proof goes as follows:

By the induction assumption of truncation induction;

$$X_3(x+1,w,i) \rightarrow (x+1 = 0 \rightarrow y = i\ ) \;\&\; (x+1 > 0 \rightarrow (y = i * (i-1) * \ldots * (x+2) \;\&\; x+1 < i))$$

Therefore $X_3(x+1,w,i) \rightarrow (x+1 > 0 \rightarrow (y = i * (i-1) * \ldots * (x+2) \;\&\; x+1 < i))$.

Therefore:

$$(\exists w)[y = w * (x+1) \;\&\; X_3(x+1,w,i) \;\&\; x+1 \neq 0] \rightarrow ((x = 0 \rightarrow y = i!) \;\&\; (x > 0 \rightarrow (y = i * (i-1) * \ldots * (x+1) \;\&\; x < i))).$$

By the truncation induction rule, we have:

$$Conv_3(\mathcal{L}_1)(\mathcal{K}) \rightarrow (x = 0 \rightarrow y = i!) \;\&\; (x > 0 \rightarrow (y = i * \ldots * (x+1) \;\&\; x < i)).$$

Therefore $Conv_7(\mathcal{L}_1)(\mathcal{K}) \rightarrow y = i!$ is valid under I_N. Q.E.D.

Note the manupulation to obtain (i) corresponds to the symbolic execution for one iteration through the loop. Therefore the induction used in this example essentially is loop induction with symbolic execution. Therefore we can regard King's symbolic execution method (15) as a special case of truncation induction. The most notable advantage of this method is that the symbolic execution implicitly carries out the proofs for X_1, X_2, X_4, X_5, X_6. Thus reduces the complexity of the logical proof drastically.

As we can see from the previous examples, use of fixed-point induction rules does not solve the problem of finding inductive assertions (or key assertions), which is the main difficulty in inductive assertion methods. But if we can

establish a suitable infinitary logic in which we can directly prove properties of $\mathrm{Conv}(\mathcal{L})(\mathcal{K})$, this difficulty will be automatically solved. Also, effective approximation of $\mathrm{Conv}(\mathcal{L})(\mathcal{K})$, which is the least upper bound of an initial sub-chain of $\mathrm{Conv}(\mathcal{L})(\mathcal{K})$, will give us a lot of information about the inductive assertions. Thus it might be worthwhile to look for some heuristic method for finding the key assertions from finite approximations of $\mathrm{Conv}(\mathcal{L})(\mathcal{K})$.

8. Conclusion

Systems of continuous formula equations were investigated and the least solutions for them were obtained in terms of sets of first order infinitary formulas. Several induction methods for proving properties of these solutions were presented.

For each program schema, a system of formula equations was associated, and the least solutions of this system of equations was defined to be the logical semantics of this program schema.

It was shown that any terminating program schema has the finite logical semantics.

Application of the induction methods for logical semantics was exhibited. Several practical program verification methods were observed to be special cases of these induction methods. Especially King's symbolic execution method was proved to be a special case of the truncation induction method.

Further research can be done in the following directions:

(i) A very restricted first order infinitary logic in which we can prove properties of $Conv(\mathcal{L})(\mathcal{X})$ will solve the difficult problem of finding key assertions in the inductive assertion method.

(ii) By computing finite approximation of $Conv(\mathcal{L})(\mathcal{X})$, we might be able to obtain key assertions heuristically.

References

(1) Tarski A. (1955) A lattice theoretic fixed-point theorem and its applications Pacific J. of Maths, Vol 5.

(2) Floyd R. (1967) Assigning meaning to programs, AMS Applied Mathematics Symposia, vol 19.

(3) McCarthy J. (1963) Towards a mathematical science of computation, Information Processing : Proc. of IFIP 62.

(4) Park D. (1969) Fix point induction and proofs of program properties, Machine Intellegence, vol 5.

(5) Scott D. (1970) Outline of a mathematical theory of computation, Proc. of 4th Princeton Conference on Information Science and Systems.

(6) Manna Z. et. al. (1972) Inductive methods for proving properties of programs, Proc. of ACM Conference on Proving Assertions about Programs.

(7) Milner R. (1972) Implementation and applications of Scott's logic for computable functions, In the same proceedings as (6).

(8) Milner R. (1973) Models of LCF, Technical Report CS-73-332, Stanford Univ.

(9) Morris H. (1973) Another recursion induction principle, CACM, vol 14 No.5.

(10) Manna Z. (1973) Program schemas, Currents in the Theory of Computing, Prentice-Hall

(11) Igarashi S. (1972) Admissibility of fixed-point induction in first order logic of typed thories, MEMO AIM-168, Stanford Univ.

(12) Kanda A. (1975) On computing the semantics of program schemas, M.Sc. thesis, Queen's Univ. at Kingston.

(13) Scott D. & Strachey C. (1971) Towards a mathematical semantics of computer languages, Proc. of Symposia on Computers and Automata, Polytechnic Institute of Brooklyn.

(14) Tennent R. (1976) The denotational semantics of programming languages, CACM vol 19, No.8

(15) King J. (1976) Symbolic execution and program tesing, CACM, vol.19, No.7.

(16) Scott D. & Tarski A. (1958) The sentential calculus with infinitary long expressions, Colloquium Mathematicum, vol.11.

(17) Shoenfield J. (1967) Mathematical logic, Addison-Wesley.