ASSIGNING MEANINGS TO PROGRAMS

INTRODUCTION

This paper attempts to provide an adequate basis for formal definitions of the meanings of programs in appropriately defined programming languages, in such a way that a rigorous standard is established for proofs about computer programs, including proofs of correctness, equivalence, and termination. The basis of our approach is the notion of an interpretation of a program: that is, an association of a proposition with each connection in the flow of control through a program, where the proposition is asserted to hold whenever that connection is taken. To prevent an interpretation from being chosen arbitrarily, a condition is imposed on each command of the program. This condition guarantees that whenever a command is reached by way of a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. Then by induction on the number of commands executed, one sees that if a program is entered by a connection whose associated proposition is then true, it will be left (if at all) by a connection whose associated proposition will be true at that time. By this means, we may prove certain properties of programs, particularly properties of the form: 'If the initial values of the program variables satisfy the relation $R_1$, the final values on completion will satisfy the relation $R_2$'. Proofs of termination are dealt with by showing that each step of a program decreases some entity which cannot decrease indefinitely.

These modes of proof of correctness and termination are not original; they are based on ideas of Perlis and Gorn, and may have made their earliest appearance in an unpublished paper by Gorn. The establishment of formal standards for proofs about programs in languages which admit assignments, transfer of control, etc., and the proposal that the semantics of a programming language may be defined independently of all processors for that language, by establishing standards of rigor for proofs about programs in the language, appear to
be novel, although McCarthy [1, 2] has done similar work for programming languages based on evaluation of recursive functions.

A semantic definition of a programming language, in our approach, is founded on a syntactic definition. It must specify which of the phrases in a syntactically correct program represent commands, and what conditions must be imposed on an interpretation in the neighborhood of each command.

We will demonstrate these notions, first on a flowchart language, then on fragments on ALGOL.

DEFINITIONS. A flowchart will be loosely defined as a directed graph with a command at each vertex, connected by edges (arrows) representing the possible passages of control between the commands. An edge is said to be an entrance to (or an exit from) the command \( c \) at vertex \( v \) if its destination (or origin) is \( v \). An interpretation \( I \) of a flowchart is a mapping of its edges on propositions. Some, but not necessarily all, of the free variables of these propositions may be variables manipulated by the program. Figure 1 gives an example of an interpretation. For any edge \( e \), the associated proposition \( I(e) \) will be called the tag of \( e \). If \( e \) is an entrance (or an exit) of a command \( c \), \( I(e) \) is said to be an antecedent (or a consequent) of \( c \).

For any command \( c \) with \( k \) entrances and \( l \) exits, we will designate the entrances to \( c \) by \( a_1, a_2, \ldots, a_k \), and the exits by \( b_1, b_2, \ldots, b_l \). We will designate the tag of \( a_i \) by \( P_i \) \((1 \leq i \leq k)\), and that of \( b_j \) by \( Q_j \) \((1 \leq j \leq l)\). Boldface letters will designate vectors formed in the natural way from the entities designated by the corresponding non-boldface letters: for example, \( \mathbf{P} \) represents \((P_1, P_2, \ldots, P_k)\).

A verification of an interpretation of a flowchart is a proof that for every command \( c \) of the flowchart, if control should enter the command by an entrance \( a_i \) with \( P_i \) true, then control must leave the command, if at all, by an exit \( b_j \) with \( Q_j \) true. A semantic definition of a particular set of command types, then, is a rule for constructing, for any command \( c \) of one of these types, a verification condition \( V_c(\mathbf{P}; \mathbf{Q}) \) on the antecedents and consequents of \( c \). This verification condition must be so constructed that a proof that the verification condition is satisfied for the antecedents and consequents of each command in a flowchart is a verification of the interpreted flowchart. That is, if the verification condition is satisfied, and if the tag of the entrance is true when the