HEHNER84
Hehner, E.C.R.; Predicative Programming Part I; CACM 27, 2 (February 1984) pp. 134-143.

This paper presents a semantics for programs where each program is considered to be a predicate in a restricted notation. The advantage of this is that a specification for a program can be written in first-order calculus and then can be refined to a program via rules of Predicate Calculus.

The author first defines what it means for a specification to be achieved: a mechanism achieves a specification if every possible observation of its behaviour satisfies the specification. In this paper, the mechanism is the computer and the observations are values of certain variables on input and then their values on output, it the mechanism terminates. Since a program can be considered a complete description of the desired observable behaviour of a computer, it can be considered a specification.

To distinguish the input and output values of a variable x, x' ("x in") denotes the vaues of x before activation and x'' ("x out") denotes the value of x after termination of the mechanism. Also v is used to denote all of the variables of a mechanism. (Note this notation has been modified slightly from the paper to be representable in ASCII.)

A specification can then be given using the mechanism's input and output variable values. If the specification consists of only input values or is a disjunction of input and output values, then if the input values achieve the specification, then the specification is satisfied even if the mechanism does not terminate. However, a boolean variable can always be added to the specification to ensure termination.

Specifications can range from the weakest, true that is achieved by any mechanism, to the strongest, false that is achieved by no mechanism. For a specification S to be achievable, a necessary condition is that

$$v'. \quad v''. S$$

i.e. for each set of input values there must exist at least one set of output values that satisfies S. If this is not the case, then a specification is said to be overdetermined.

The paper then gives some definitions of programming language features and properties such as assingment, choice, local variable declaration, and program and subroutine definition to indicate the approach taken by this semantics.

The advantage of using Predicate Calculus for this semantics over a higher order calculus is simplicity. As the author points out, the real test of any semantics is the help it provides to a programmer who wishes to be rigorous. Further, this semantics is flexible because is allows the mixing of programming and logical connectives so that programs and specifications are on the same level.

In conclusion, this paper presents a semantics that by the use of Predicate Calculus achieves neatness and simplicity in its notation and yet supplies flexibility and rigour.