1. INTRODUCTION

The present paper is a progress report about our work on semantics and proof theory of programming languages. We study a number of fundamental programming concepts occurring e.g. in the language PASCAL, viz. assignment, sequential composition, conditionals, locality, and (recursive) procedures with parameters called-by-value and called-by-variable. Our goal is the development of a formalism which satisfies two requirements:

- Semantic adequacy: the definitions capture exactly the meaning attributed to these concepts in the PASCAL report.
- Mathematical adequacy: The definitions are as precise and mathematically rigorous as possible.

Of course, full semantic adequacy cannot be achieved within the scope of our paper. Thus, we were forced to omit certain aspects of the concepts concerned. What we hope to have avoided, however, is any essential alteration of a concept for the sake of making it more amenable to formal treatment.

Our approach follows the method of denotational semantics introduced by Scott and Strachey (e.g. in [12]). Moreover, we investigate the connections between denotational semantics and Hoare's proof theory ([6]), insofar as pertaining to the concepts mentioned above.

As main contributions of our paper we see:

- The proposal of a new definition of substitution for a subscripted variable. This allows an extension of Hoare's axiom for assignment to the case of assignment to a subscripted variable. (This idea is described in greater detail in [2].)
- The proposal of a semantic definition and corresponding proof rule for recursive procedures with an adequate treatment of call-by-value and call-by-variable. (We believe these to be new. The proof rule is based on Scott's (or computational) induction, which is well-understood for parameterless procedures, but hardly so for procedures with parameters. In our opinion, neither the papers of Manna et al. (e.g. in [10, 11]) nor those of e.g. De Bakker ([1]), Hoare ([7]), Hoare and Wirth ([8]), Igarashi, London and Luckham ([9]) give the full story on this subject.)

It will turn out that our treatment of procedures is quite complex. However, we doubt whether an approach which is essentially simpler is possible. Of course, we do not claim that our formalism is the last word, but the programming notions involved are intricate,
and we feel that essential simplification could be obtained only by changing the lan-
guage.

The paper has the following outline:

Section 2 gives the syntax of the various language constructs. Also, a careful defini-
tion of substitution is given which is needed for the treatment of assignment, local-
ity and parameter passing.

Section 3 is devoted to the definition of the denotational semantics of the five types
of statements. We introduce the semantic function $M$ which gives meaning to a statement
$S$, in a given environment $e$ (a mapping from variables to addresses) and store $σ$ (a map-
ping from addresses to values), yielding a new store $σ': M(S)(e, σ) = σ'$. For assign-
ment, sequential composition and conditionals the definitions are fairly straightfor-
ward. It is also reasonably clear what to do about locality, but the treatment of pro-
cedures may be rather hard to follow. Some of the causes are:

- When applying the usual least fixed point approach, one has to be careful with the
types (in the set-theoretical sense) of the functions involved.
- The notion of call-by-variable (the FORTRAN call-by-reference) requires a somewhat
mixed action to be taken: When the actual parameter (which has to be a variable) is
subscripted, the subscript is evaluated first, and then a process of substitution of
the modified actual for the formal is invoked.
- The possibility of clash of variables has to be faced. (Cf. the ALGOL 60 report,
sections 4.7.3.2 (Example: $b$ int $x$; proc $P(x)$; int $x$; $b$ int $x$; $e$; ...$P(x+1)$; ...$e$) and 4.7.3.3
(Example: $b$ int $x$; proc $P$; $b$ int $x$; $e$; ...$b$ int $x$; ...$P$; ...$e$; ...$e$.) These problems are not
exactly the same as encountered in mathematical logic; in particular, they cannot
simply be solved by appropriate use of the notions of free and bound occurrence and
of substitution, as customary in logic.

Section 4 introduces the proof-theoretical framework. It contains the "Exercises in
denotational semantics": For each type of statement, a corresponding axiom or proof
rule is given, and it is required to show its soundness. Also, a modest attempt at
dealing with substitution is included. In fact, for two rules (sequential composition
and conditionals) the proof is easy, for the assignment axiom we refer to [2], whereas
the remaining three cases should, at the moment of writing this, be seen as conjectures
since we do not yet have fully worked out proofs available. However, we are confident
that the rules, perhaps after some minor modifications, will turn out to be sound.

It may be appropriate to add an indication of the restrictions we have imposed
upon our investigation. There are a few minor points (such as: only one procedure de-
claration, i.e., not a simultaneous system; only one parameter of each of the two types,
etc.). Next, things we omitted but which we do not consider essentially difficult (such
as type information in declarations) and, finally, a major omission: We have no func-
tion designators in expressions, nor do we allow procedure identifiers as parameters.

There is a vast amount of literature dealing with the same issues. Many of the
papers take an operational approach, defining semantics in terms of abstract machines.