0. ABSTRACT

This paper presents a formulation for the standard semantics of block structure and ALGOL 60 style call-by-name. The main features of this formulation are the use of continuations and streams. Continuations are used in such a way that the semantics can be defined without requiring the idea of an explicit store. Thus the concepts of address or L- and R-values are not used, and simple continuations suffice for describing assignments, iterative control statements, compounds, blocks, and functions using call by value. (Side effects are still allowed via assignments to variables global to functions.) Call-by-name is handled by introducing the idea of multiple continuations. Input-output is treated by using streams. In conjunction with continuations, these allow the formulation of program "pipes" exactly like compound functions.
1. Introduction

The purpose of denotational semantics, according to Milne and Strachey [8], is to provide an accurate standard by which designers and implementors of programming languages can judge their work. That standard must not be any more specific than is necessary, lest the definition of the language enforce too strong a constraint on its implementation; with new hardware and architectures becoming available one can easily foresee that an overly specific definition will preclude growth of software onto those new machines. "One singularly elegant sort of semantics, 'standard semantics,' is of special significance," because "it reduces to a minimum the amount of substantial information" that is manipulated; "whenever possible precedence should be given to standard semantics rather than store semantics." [8, pp.11-12] The semantics of a programming language should "be specified in the first instance by using standard semantics."

Following the spirit of that dictum, we here demonstrate how to avoid the store entirely in providing semantics for the archetype of languages that seem to require it. ALGOL 60 was designed before development of the tools now familiar to denotational semanticists, but it was designed so carefully [9] that the need for such tools became obvious. By substantially solving the problem of specifying syntax precisely, its designers hastened the development of formal semantics [7].

It was designed, however, with the traditional store in mind. Thus, we believe, much of the early formal semantics also presumed the necessity of that structure. Here we show again [1], but more clearly, how the store might be avoided entirely without changing the (understood) meaning of ALGOL.

We are not the only ones doing this; Brookes attacks the same problem elsewhere in this volume [2]. His approach is beautiful, though abstract; ours is effective. To the extent that the lambda calculus is operational, one might say that ours is closer to an implementation; his is clean and elegant. We feel, however, that a simple implementation in the lambda calculus (and, except for the strange machinations of call-by-name, this is simple!) will be sufficiently abstract to generalize to almost any machine [5]. Indeed, we also believe that our semantics is 'fully abstract.'

Previous efforts to define most of the programming features in languages like ALGOL 60 [e.g., 5, 8] pivot on store semantics, with the store as one domain that maps the so-called L-values (locations) to R-values (contents). This domain has been deemed necessary because even such fundamental features as program vari-