THE SEMANTICS OF NONDETERMINISM

by

T.S.E. Maibaum

Research Report CS-77-30

Department of Computer Science

University of Waterloo
Waterloo, Ontario, Canada

December 1977

§0 Introduction

One of the most intriguing topics in Mathematical semantics in the last few years has been that of non-determinism. Although very few existing languages allow non-determinism, the study of such languages is not without merit. For example, any language which deals with relations as opposed to functions (e.g. query languages for relational data bases) must be in some way nondeterministic. In [4], Dijkstra has introduced a non-deterministic language which he claims facilitates the synthesis of programs. Moreover, many authours have studied parallelism by using the concept of non-determinism.

Non-determinism means, of course, allowing some element of chance to influence how a computation might proceed. As a first approach, we might introduce a choice construct "or" into a simple language of recursive definitions. As in [17], these recursive definitions give rise to evaluation sequences and the application of the evaluation mechanism to a "program segment" T_1 or T_2 would result in a random choice to evaluate either T_1 or to evaluate T_2.

The relevant domains of interpretation for these recursive definitions are non-deterministic domains or structures which are special instances of a class of domains suggested in [5]. The elements of this idea appear in [13] and [6] and were formally pointed out in [8], [9]. A structure is an element of a restricted class of complete partially ordered sets (cpo's). The restriction is a consequence of requiring that we not only have the usual so-called "computational partial order" ($\sqsubseteq$) on data domains, we also order domains by a so-called "results partial order".

The reasoning behind the choice of this restricted class is explained as follows: We assume that our machine is equipped with basic functions which are deterministic (i.e. return at most one output when given

some input). The non-determinism results from having a choice construct in a programming language. Any given execution of a nondeterministic program P will result in a deterministic computation. However, many different computations may be executions of P and these computations (call them C_p) may or may not be comparable using the usual ordering of computations. The result of executing P could be the output of any of these computations. C_p. What could be the output of an execution of P_1 or P_2? The output could be an output of an execution of P_1 or an output of an execution of P_2. Thus (informally) result $(P_1$ or $P_2)$ = join (result (P_1), result (P_2)) where join: (sets of results)^2 → (sets of results). Moreover, even if the computations of P_1 and P_2 (on the same input) are not comparable (using the "computational partial order"), we may be able to show that result (P_1) approximates result (P_2) with respect to the join operation indicated above.

Another important problem in studying computations is how to construct function spaces of given domains. For example, if D is a cpo, then $[D \to D]$ is the set of continuous functions from D to D and is easily shown to be a cpo. The fact that D and $[D \to D]$ have similar properties as domains is vital in studying deterministic computations. Since we restrict the class of cpo's we may use in studying nondeterministic computations, do we also need to restrict the class of functions we allow in order to maintain these special properties? The answer is of course in the affirmative: given a structure D, we let $[D,D]$ be the class of functions which are continuous with respect to the computational partial order and monotonic with respect to the results partial order. This reflects the intuitive idea that if we give "more" inputs to a nondeterministic program, then we should expect "more" outputs.

As to the contents of the paper, in Section 1, we outline some underlying mathematical ideas. In Section 2, we study the class of structures and show that there is a universal structure; that is, we show that there is a domain in which nondeterministic programs can be given meaning symbolically and that interpretations of this symbolic meaning in other structures are consistent with the meaning of these programs in these structures. In Section 3, we show that these ideas can be generalised to give definitions of nondeterministic programs of higher type: i.e. nondeterministic functionals.