ABSTRACT

Functional Programming (FP) systems are modified and extended to form Nondeterministic Functional Programming (NFP) systems in which nondeterministic programs can be specified and both deterministic and nondeterministic programs can be verified essentially within the system. It is shown that the algebra of NFP programs has simpler laws in comparison with the algebra of FP programs. "Regular" forms are introduced to put forward a disciplined way of reasoning about programs. Finally, an alternative definition of "linear" forms is proposed for reasoning about recursively defined programs. This definition, when used to test the linearity of forms, results in simpler verification conditions than those generated by the original definition of linear forms.

1. INTRODUCTION

In [1] Backus introduced a class of applicative programming systems called FP (Functional Programming) systems as an alternative to the conventional style of programming. All programs in an FP system are deterministic, and represent strict functions over some flat domain D of objects. Each FP system associates with it a finitely-generated algebra (hereinafter called FP algebra) \(<P,F>\) where P is the set of programs and \(F\) is a set of continuous functionals. A finite set of primitive programs is the generating set of the algebra. A remarkable aspect of the FP systems is that the rules of the FP algebra can be specified as a collection of simple laws and theorems based on functional identities. These laws and theorems can be used to reason about programs by transformations. An FP system also allows recursive definitions of programs. Algebraic methods for reasoning about recursively defined programs of several types are given in [1,2,5,6].

Any reasonably powerful FP system should have the "condition" functional to provide the programmer with a facility to define branching computations. With the boolean constants \(T\) and \(F\) (having usual meanings) as objects, this functional is defined by:
Definition 1.1

For all programs p, q, r, and for all objects x:

\[(p\cdot q; r): x = q: x \text{ if } p: x = T,\]
\[r: x \text{ if } p: x = F,\]
\[\bot \text{ otherwise,}\]

where \(\bot\) is the "undefined" object (the least element of D).

The laws involving condition and other functionals are unduly complicated in the sense that any one branch of the condition reflects the essential characteristics of the laws. Also, programs requiring more than two alternative branches have to be simulated in a round-about way by nested conditions. Complexity involved in the undue abstraction offered by the condition is also encountered in the study of "linear" forms (forms are FP program schemas), which have been introduced by Backus to reason about recursively defined programs [2]. Backus has proposed the following definition of linear forms:

Definition 1.2

A form \(H(f)\) (in the program variable \(f\)) is linear if and only if there is a form \(H_\tau(f)\), called the predicate transformer of \(H\), such that

1. For all programs p, q, and r,
\[H(p\cdot q; r) = H_\tau(p) \cdot H(q); H(r),\]

and

2. For all objects x, and for all programs p,
\[H(\bot) : x \neq \bot \Rightarrow H_\tau(p) : x = T\]

where \(\bot\) is the program such that \(\bot: x = \bot\) for all objects x.

This definition cannot always be successful as a test to determine whether a form \(H\) is linear because, if \(H\) is not linear, it cannot guarantee that \(H\) does not have a predicate transformer. Even if a form \(H\) is linear, the necessity of transforming \(H(p\cdot q; r)\) into the form \(H_\tau(p) \cdot H(q); H(r)\) by applying the laws of FP algebra makes the test sometimes difficult, especially if \(H\) has a complex structure. In [2] Backus has started a study of linear forms to find the relation between linearity and structure of the forms. The results of this study, as have been obtained so far, are useful, because a form can be shown linear (or, nonlinear) by examining the linearity of the components of the form. The involved object level reasoning necessitated by the aforesaid definition of linear forms tends to further complicate the study of linear forms.

In this paper we attempt to find solutions of these problems.