Science of Computer Programming 4 (1984) 141–157
North-Holland

PARTIAL CORRECTNESS: THE TERM-WISE APPROACH

Stefan SOKOŁOWSKI
Institute of Mathematics, University of Gdańsk, ul. Wita Stwosza 57, 80–952 Gdańsk, Poland

Communicated by D. Bjørner
Received June 1982
Revised November 1983

1. Introduction

The need for new notions of partial correctness has emerged from studies on cooperation of applicative and imperative elements of programs. Whereas the latter can be neatly specified and verified by the usual inductive assertions, there is no apparent way to apply the same approach to the former. The obvious reason is that functions are not called in order to change states—the realm that assertions can capture. On the other hand, while the lambda-calculus is perfectly suited to functions and values, its treatment of imperative elements of programs is hardly satisfactory.

This paper puts forward a natural way to incorporate functions into a programming logic built along the lines of Hoare's axiom system (see [6]); which we call *term-wise correctness.*

The first reason to consider the correctness of programs with respect to terms rather than predicates is a simple observation that the classical Hoare's logic, although complete, is too weak to *specify* programs. Even if we know that a command c is correct with respect to predicates

$$n \geqslant 0 \quad \text{and} \quad x = n!$$

we still cannot claim that c computes factorials since c might update n instead (e.g. c might be: $n := 0$; $x := 1$). One way out is to carefully distinguish between variables and constants. Another is Manna's and Pnueli's binary postassertions method (see [8]). Still another, put forward in this paper, is to allow

$$\{t_1\}\, c\, \{t_2\}$$

where t_1 and t_2 are terms and c is a command, to mean

the value of t_2 after execution of c is,
if defined, equal to the value of t_1 before

Now $\{n!\}\, c\, \{x\}$ means that c assigns to x the factorial of the initial value of n.

0167-6423/84/$3.00 © 1984, Elsevier Science Publishers B.V. (North-Holland)

Although the idea behind this new notion of partial correctness is so different, the actual correctness calculus is very close to the classical Hoare's logic. Backward substitution and loop invariants work in a standard way—but now, expressions are substituted for variables in terms rather than in predicates and loop invariants are terms rather than predicates.

In the world of recursive procedures and functions the classical Hoare's logic, with the imposed distinction between constants and variables, becomes confusing. An assertion inside a procedure body that relates a local variable to a constant should at the same time relate other incarnations of the same variable on different levels of recursion to other constants. Presumably the simplest treatment is the one described by Apt in [1]. For recursive procedures with local variables one needs 6 additional inference rules and 1 axiom.

O'Donnell in [9] gave a critical study of existing inference systems for user defined functions. O'Donnell's conclusion is that all known inference systems (including one of his own) are either unsound or impractical in that they do not allow for a correctness proof to reflect the structure of the user's program.

I believe that the inference system presented here escapes O'Donnell's criticism. It is, moreover, relatively simple. It contains at most one rule for each syntactic construct plus three consequence rules; the proofs of correctness do not involve such unwelcome concepts as explicit locations, stacks for procedures and environments. It can deal conveniently with local variables and with parameters called either by value or by value-result. The inference system is provably sound and Cook-complete (cf. [3]). However, not to leave the false impression that this system has the best of all worlds, it cannot treat nested declarations of procedures, procedural parameters, side effects and pointers.

2. Algebra of partial functions

On the semantic level, all elements of programs and their descriptions boil down to partial functions: commands yield functions from states to states; expressions, from states to values; procedures, from tuples of values to tuples of values; and assertions, from states to logical values. If we can express partial correctness in a simple theory of partial functions, the applicative elements will fit there as well.

Fundamental operations over partial functions are composition, tupling, restriction, choice and iteration.

Composition of functions f and g is defined by

$$(f\circ g)(a)=\begin{cases} g(f(a)) & \text{if } f(a) \text{ is defined,}\\ \text{undefined} & \text{otherwise}\end{cases}$$

(note that f is applied first).

Tupling of partial functions $f_1,\ldots,f_n$ is a function into the Cartesian product of their destinations defined as follows:

Download English Version:

https://daneshyari.com/en/article/6700395

Download Persian Version:

https://daneshyari.com/article/6700395

Daneshyari.com