Calculating with procedure calls

A. Bijlsma
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, The Netherlands

April 8, 1993

Keywords Formal semantics, program correctness, program specification, procedures.

1 Introduction

Although various proof rules for procedure calls in weakest precondition semantics have been proposed over the years [3, chapter 12] [4] [1], none of these is particularly suitable for calculational program construction. The problem is that they tend to yield a precondition that is tortuously expressed. For instance, let a procedure for finding the integer square root of a parameter y be specified, using a specification variable m, by the precondition

$$m^2 \leq y < (m+1)^2$$

and the postcondition

$$y = m \quad .$$

Suppose this procedure is called with argument b and we wish to find a precondition such that after the call $b < 5$ holds. Then the proof rule from [1] tells us that such a precondition is given by

$$\begin{array}{l}(\exists m :: m^2 \leq b < (m+1)^2) \\ \wedge\ (\forall y :: (\forall m :: m^2 \leq b < (m+1)^2 \ \Rightarrow\ y = m) \ \Rightarrow\ b < 5) \quad .\end{array}$$

Although this is certainly equivalent to the desired answer $0 \leq b < 25$, proving so requires a certain virtuosity in the use of predicate calculus.

The cause of this difficulty is the desire to create a proof rule that gives an answer of maximal precision for every specification, no matter how awkwardly the latter may be expressed. In this paper our approach is different: we give a proof rule that is very easy to use, but does not yield the best possible answer for every specification—only for a large class of specifications that we shall call *normal specifications*. In the appendix we show how every specification can be transformed into an equivalent normal one; since this labour must be done only once for every procedure (as opposed to once for every procedure call), it seems sensible to make it the responsibility of the procedure designer and require all specifications to be normal in the first place. In practice, they already are: it is our experience that non-normal specifications occur only in pathological cases expressly designed as counterexamples.

2 The proof rule

In order to study procedures with value parameter x, value-result parameter y, and result parameter z, we let $\mathcal{B}$ denote the state space consisting of x, y, and z only, and we define a *procedure body* S to be a statement over $\mathcal{B}$ such that

$$(\forall v :: [x = v \;\Rightarrow\; wlp.S.(x = v)]) \quad , \tag{1}$$

$$(\forall R :: (\forall v :: [(z := v).(wp.S.R) \;\equiv\; wp.S.R])) \quad . \tag{2}$$

Condition (1) is meant to model that S does not change the value of x, condition (2) that S does not depend on the value of z. The following notational conventions have been used: square brackets denote quantification over variables, v ranges over values and R over predicates; the predicate $(z := v).R$ is obtained from R by replacing all free occurrences of z by v.

A *procedure specification* is a pair (U, V) of predicates in the variables of $\mathcal{B}$ and an additional specification variable m such that U is independent of z. A procedure body S is said to *satisfy* the specification (U, V) if

$$[U \;\Rightarrow\; wp.S.V] \quad . \tag{3}$$

Let $\mathcal{C}$ be a state space disjoint from $\mathcal{B}$ and not containing m. Let a be an expression over $\mathcal{C}$, and let b and c be distinct variables of $\mathcal{C}$. For S a procedure body, the *procedure call* $S(a, b, c)$ is defined to be equivalent to the following sequence of statements:

$$x, y := a, b; \quad S; \quad b, c := y, z \quad . \tag{4}$$

Let (U, V) be a given specification. A *proof rule* for procedure calls prescribes, for every predicate E over $\mathcal{C}$ (the call's postcondition), a predicate D over $\mathcal{C}$ (its precondition) with the property that, for all procedure bodies S that satisfy (U, V),

$$[D \;\Rightarrow\; wp.S(a, b, c).E] \quad . \tag{5}$$

The rule is called *sharp* if D is the weakest such predicate [1]. If (U, V) is not satisfied by at least one body, the weakest D is always *true*, independently of E. From now on, we assume that (U, V) is satisfied by at least one body.

The proof rule that we study in the present paper is contained in the following theorem. To simplify its formulation and proof, we introduce the abbreviations

$$U' \quad \text{for} \quad (x, y := a, b).U \quad , \tag{6}$$

$$V' \quad \text{for} \quad (x := a).V \quad , \tag{7}$$

$$E' \quad \text{for} \quad (b, c := y, z).E \quad . \tag{8}$$

Remembering which substitutions are to be performed here is made easier if the reader keeps in mind the form of (4) and the axiom of assignment.

Theorem 1 *Any predicate D over $\mathcal{C}$ satisfying both*

$$[U' \wedge V' \;\Rightarrow\; (D \Rightarrow E')] \tag{9}$$

and

$$[D \;\Rightarrow\; (\exists m :: U')] \tag{10}$$

is a solution of (5).

Proof We may rewrite (9) according to

$$
\begin{array}{ll}
 & [U' \wedge V' \;\Rightarrow\; (D \Rightarrow E')] \\
\equiv & \quad \{\text{shunting}\} \\
 & [D \;\Rightarrow\; (U' \wedge V' \Rightarrow E')] \\
\equiv & \quad \{\text{distribution, using } D \text{ independent of } y, z, m\,\} \\
 & [D \;\Rightarrow\; (\forall y, z, m :: U' \wedge V' \Rightarrow E')] \\
\equiv & \quad \{\text{distribution, using } E' \text{ independent of } m\,\} \\
 & [D \;\Rightarrow\; (\forall y, z :: (\exists m :: U' \wedge V') \Rightarrow E')] \\
\equiv & \quad \{\text{trading}\} \\
 & [D \;\Rightarrow\; (\forall y, z :: (\exists m : U' : V') \Rightarrow E')] \quad .
\end{array}
$$

Hence the conjunction of (9) and (10) is equivalent to

$$[D \;\Rightarrow\; (\exists m :: U') \;\wedge\; (\forall y, z :: (\exists m : U' : V') \Rightarrow E')] \quad . \tag{11}$$

Compare this with the proof rule from [1], which reads

$$[D \;\Rightarrow\; (\exists m :: U') \;\wedge\; (\forall y, z :: (\forall m : U' : V') \Rightarrow E')] \quad . \tag{12}$$

To prove $(12) \Leftarrow (11)$, it is sufficient to show that

$$[(\exists m :: U') \;\Rightarrow\; ((\forall m : U' : V') \Rightarrow (\exists m : U' : V'))] \quad ,$$

which is elementary predicate calculus (see e.g. Theorem (7,34) of [2]).
□

We prefer to work with (9) and (10) rather than (11) or (12) for two reasons. First, verification of (10) tends to be a triviality in practical examples, since (10) merely states that the arguments lie within the procedure's domain of applicability and has nothing to do with the desired postcondition E. Secondly, the calculation of D satisfying (9) can be made the subject of a formal derivation of the shape

$$
\begin{array}{ll}
 & E' \\
\Leftarrow & \quad \{\, U' \wedge V' \,\} \\
 & \cdots \\
\Leftarrow & \quad \{\, U' \wedge V' \,\} \\
 & D
\end{array}
$$

Thus D is obtained by a strengthening transformation of E' into a predicate over $\mathcal{C}$ under assumption of the specification predicates U' and V', much like the precondition of an assignment is simplified under assumption of an invariant. Due to the absence of explicit quantification, such a derivation leads to much tidier calculations than those resulting from an explicit formula like (12). In Section 4, we give some examples to illustrate this. An additional benefit, less obvious from these small examples, is that, in the case where U' and V' consist of many conjuncts, the suggested proof format gives the opportunity of using these conjuncts one at a time, the width of the derivation meanwhile being determined by that of the desired postcondition E only.

3 Sharpness

Comparing (11) with (12), which is sharp, shows that (11) is also sharp if we can guarantee

$$[(\exists m : U' : V') \;\Rightarrow\; (\forall m : U' : V')] \quad .$$

By simple predicate calculus (e.g. Theorem (7,33) of [2]) we find the sufficient condition

$$[(\mathsf{N}\, m :: U) \leq 1] \quad .$$

That is why we define

Definition 2 A specification (U, V) is called *normal* if

$$[(\mathsf{N}\, m :: U) \leq 1]$$

holds.
□

Then the proof rule from Theorem 1 is sharp for normal specifications.

The special case where U and V mention no specification variables is absorbed by letting the type of m be a singleton set; hence specifications without specification variables are normal.

4 Examples

Example 3 Consider again the example mentioned in the Introduction, viz. the specification

$$\begin{array}{ll} \text{pre} & U: \quad m^2 \leq y < (m+1)^2 \quad , \\ \text{post} & V: \quad y = m \quad . \end{array}$$

The domain condition, i.e. the consequent of (10), is

$$(\exists m :: m^2 \leq b < (m+1)^2)$$

or, equivalently, $0 \leq b$. The part of the derivation corresponding to the postcondition $b < 5$ is

$$\begin{array}{ll} & y < 5 \\ \equiv & \quad \{\, V': \quad y = m \,\} \\ & m < 5 \\ \equiv & \quad \{\, U': \quad m^2 \leq b < (m+1)^2 \,\} \\ & b < 25 \quad . \end{array}$$

Hence the desired precondition is $0 \leq b < 25$.
□

Example 4 This example will show what happens when the value argument a and the result argument c coincide. That is to say, we consider a call $S(c, c)$, where S is specified by

$$\begin{array}{ll} \text{pre} & U: \quad \mathit{true} \quad , \\ \text{post} & V: \quad z = x + 1 \quad . \end{array}$$

Now the domain condition is *true*. The postcondition $c < 5$ gives

$$
\begin{array}{ll}
 & z < 5 \\
\equiv & \quad \{\, V' : \quad z = c + 1 \,\} \\
 & c + 1 < 5 \\
\equiv & \quad \{\} \\
 & c < 4 \quad .
\end{array}
$$

So the desired precondition is $c < 4$. (Notice that this example is disallowed by many proof rules.)

□

Example 5 Here is an example with two value-result parameters $y0, y1$; the procedure is supposed to swap their values. Using two specification variables $m0, m1$, we may express this as

$$
\begin{array}{ll}
\text{pre} & U : \quad y0 = m0 \ \wedge \ y1 = m1 \quad , \\
\text{post} & V : \quad y0 = m1 \ \wedge \ y1 = m0 \quad .
\end{array}
$$

The domain condition is again *true*. Consider a call $S(b0, b1)$ with postcondition, say, $b0 < b1 < b2$, where $b0, b1, b2$ are distinct variables. This gives

$$
\begin{array}{ll}
 & y0 < y1 < b2 \\
\equiv & \quad \{\, V' : \quad y0 = m1 \ \wedge \ y1 = m0 \,\} \\
 & m1 < m0 < b2 \\
\equiv & \quad \{\, U' : \quad b0 = m0 \ \wedge \ b1 = m1 \,\} \\
 & b1 < b0 < b2 \quad .
\end{array}
$$

The desired precondition is $b1 < b0 < b2$.

□

The reader is invited to check that each of the above specifications is normal. It follows that we have obtained the best possible preconditions.

Appendix: Normalizing specifications

Theorem 6 *Every specification is equivalent to a normal specification.*

Proof Consider a specification (U, V). Let μ denote a fresh specification variable. We claim that

$$
\begin{array}{ll}
\text{pre} & Q : \quad y = \mu \ \wedge \ (\exists m :: U) \quad , \\
\text{post} & R : \quad (\forall m :: (y := \mu).U \ \Rightarrow \ V)
\end{array}
$$

is a normal specification equivalent to (U, V).

Normality is easy, since $[(\mathsf{N}\ \mu :: y = \mu) = 1]$. To prove equivalence, we consider both implications separately.

Assume that S satisfies (U, V). Then we have, for all μ,

$wp.S.((y := \mu).U \ \Rightarrow \ V)$
$\equiv$ {elimination of $\Rightarrow$}
$wp.S.(\neg(y := \mu).U \ \vee \ V)$
$\Leftarrow$ {monotonicity of wp}
$wp.S.(\neg(y := \mu).U) \ \vee \ wp.S.V$
$\equiv$ {Transparency Lemma [1], using $\neg(y := \mu).U$ independent of y, z}
$(\neg(y := \mu).U \ \wedge \ wp.S.true) \ \vee \ wp.S.V$
$\Leftarrow$ {monotonicity of wp}
$(\neg(y := \mu).U \ \wedge \ (\exists m :: wp.S.V)) \ \vee \ wp.S.V$
$\Leftarrow$ { $[U \ \Rightarrow \ wp.S.V]$ }
$(\neg(y := \mu).U \ \wedge \ (\exists m :: U)) \ \vee \ U$,

so

$[Q \ \Rightarrow \ wp.S.R]$
$\equiv$ {definitions of Q and R}
$[y = \mu \ \wedge \ (\exists m :: U) \ \Rightarrow \ wp.S.(\forall m :: (y := \mu).U \ \Rightarrow \ V)]$
$\equiv$ {universal conjunctivity of wp}
$[y = \mu \ \wedge \ (\exists m :: U) \ \Rightarrow \ (\forall m :: wp.S.((y := \mu).U \ \Rightarrow \ V))]$
$\equiv$ {antecedent independent of m}
$[y = \mu \ \wedge \ (\exists m :: U) \ \Rightarrow \ wp.S.((y := \mu).U \ \Rightarrow \ V)]$
$\Leftarrow$ {previous derivation}
$[y = \mu \ \wedge \ (\exists m :: U) \ \Rightarrow \ (\neg(y := \mu).U \ \wedge \ (\exists m :: U)) \ \vee \ U]$
$\equiv$ {elimination of μ by the one-point rule}
$[(\exists m :: U) \ \Rightarrow \ (\neg U \ \wedge \ (\exists m :: U)) \ \vee \ U]$
$\equiv$ {complement rule}
$[(\exists m :: U) \ \Rightarrow \ (\exists m :: U) \ \vee \ U]$
$\equiv$ {propositional calculus}
$true$,

which shows that S satisfies (Q, R).

For the other direction, we assume that S satisfies (Q, R), i.e.,

$$[y = \mu \ \wedge \ (\exists m :: U) \ \Rightarrow \ wp.S.(\forall m :: (y := \mu).U \ \Rightarrow \ V)] \tag{13}$$

and we derive, for μ to be determined in the course of the derivation,

$wp.S.V$
$\Leftarrow$ {monotonicity of wp}
$wp.S.((y := \mu).U \ \wedge \ V)$
$\equiv$ {propositional calculus}
$wp.S.((y := \mu).U \ \wedge \ ((y := \mu).U \ \Rightarrow \ V))$
$\equiv$ {conjunctivity of wp}
$wp.S.((y := \mu).U) \ \wedge \ wp.S.((y := \mu).U \ \Rightarrow \ V)$
$\equiv$ {Transparency Lemma [1], using $(y := \mu).U$ independent of y, z}
$(y := \mu).U \ \wedge \ wp.S.true \ \wedge \ wp.S.((y := \mu).U \ \Rightarrow \ V)$
$\equiv$ {monotonicity of wp}
$(y := \mu).U \ \wedge \ wp.S.((y := \mu).U \ \Rightarrow \ V)$
$\Leftarrow$ {monotonicity of wp}

$$
\begin{array}{ll}
 & (y := \mu).U \ \wedge \ wp.S.(\forall m :: (y := \mu).U \ \Rightarrow \ V) \\
\Leftarrow & \quad \{(13)\} \\
 & (y := \mu).U \ \wedge \ y = \mu \ \wedge \ (\exists m :: U) \\
\equiv & \quad \{\text{in particular, we now choose } \mu \text{ to be } y\,\} \\
 & U \ \wedge \ (\exists m :: U) \\
\equiv & \quad \{\text{instantiation}\} \\
 & U \ .
\end{array}
$$

□

References

[1] A. Bijlsma, P.A. Matthews, and J.G. Wiltink. A sharp proof rule for procedures in wp semantics. *Acta Inf.* **26** (1989), 409–419.

[2] E.W. Dijkstra & C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer, New York, 1990.

[3] D. Gries. *The Science of Programming*. Springer, New York, 1981.

[4] A.J. Martin. A general proof rule for procedures in predicate transformer semantics. *Acta Inf.* **20** (1983), 301–313.