A Correctness Proof of Sorting by means of Formal Procedures

Maarten M. Fokkinga
University of Twente, Faculty of Informatics
P.O. Box 217, NL 7500 AE ENSCHEDE
Netherlands

Reprint from: Science of Computer Programming 9 (1987) 263–269

Abstract. We consider a recursive sorting algorithm in which, in each invocation, a new variable and a new procedure (using the variable globally) are defined and the procedure is passed to recursive calls. This algorithm is proved correct with Hoare-style pre- and postassertions. We also discuss the same algorithm expressed as a functional program.

1. Introduction

Proving the correctness of programs using Hoare-style assertions [4], is nowadays a well-known and well-used method; see e.g. any modern textbook on programming. The applications, however, rarely involve programs with "formal procedures", i.e. procedures as parameters. Yet the use of formal procedures forms a powerful and intriguing programming technique, especially in combination with recursion. Recently Van Eijk [2] gives a useful example and Augusteijn [1] illustrates this programming technique by a wide variety of examples in Pascal. Earlier, Kruseman Aretz [5] has used Algol 60's *name* parameter mechanism instead of formal procedures. Actually, arbitrary Turing machines and similar devices can be simulated by procedures and recursion only (and without other data types like *integer* and *boolean*), as shown by Langmaack [6,7,8] and Fokkinga [3]. (Of course, for anyone familiar with the Lambda calculus this comes as no surprise, but what is important here is that procedures as result are not needed in these programs.)

In this note we consider a sorting program due to Matthijs Kuiper (State University of Utrecht, NL), that uses a locally declared variable v in a recursive procedure p to store the values to be sorted; a local procedure q, declared together with v, is passed to recursive calls of p and provides access to v while it is held on the stack of incarnations of p. The program hardly admits an operational explanation (in terms of machine actions and stored values) without getting imprecise or sloppy. The correctness, however, is easily shown by Hoare-style assertions and the annotation of the program is in our opinion a clear and convincing explanation of its working.

2. The Program

We use *in* and *out* as the variables containing the input and output respectively. The program is to read values from the input and to write them in increasing order (hence without duplicates) on the output. Here is the program, in a Pascal-like notation.

```
proc p (proc q (int, int))
      = var v : int;
        proc q1 (m, n : int)
              = begin  if m < v < n
                       then q(m, v); write(out, v); q(v, n)
                       else q(m, n)
                       fi
                end;
        begin  if in = ∅
                 then q(−∞, +∞)
                 else read(in, v);
                      p(q1)
                 fi
        end;
proc q0 (m, n : int)
      = begin skip end;
begin p(q0) end
```

We shall give a correctness proof with the well-known assertion method. The remarkable thing is that not only "conventional" assertions are needed, like $in = S$, but also "unconventional" assertions like $\forall\, T, m, n.\ \{P\}\ q(m, n)\ \{Q\}$ as *part of* an assertion (asserting something about procedure parameter q).

3. The Correctness Specification

We shall first formulate the specification of p; the correctness will be shown in the following section. The specification is sufficiently strong to show the correctness of the body of p. For the application of p in the main program, i.e. the call $p(q0)$, the specification could have been weaker. So, because the specification is strong enough for the correctness proof of the body, it forms the heart of the explanation (documentation) of the working of p. Here it is (with some explanation following it).

$$pspec(p) \quad = \quad (\forall\, R, S, q.\ \{P1\ \wedge\ qspec(R, q)\}\ p(q)\ \{P2\})$$

where

$$\begin{array}{lcl} P1 & \equiv & in = S\ \wedge\ out = \varnothing \\ P2 & \equiv & in = \varnothing\ \wedge\ out = sort(-\infty, R +\!\!+ S, +\infty) \\ qspec(R, q) & = & (\forall\, T, m, n.\ \{out = T\}\ q(m, n)\ \{out = T +\!\!+ sort(m, R, n)\}) \\ sort(m, R, n) & = & \text{the increasing sequence of all values } x \text{ from } R \text{ for which } m{<}x{<}n \end{array}$$

Throughout this note we let R, S and T vary over sequences; R denoting the part already Read in, S denoting the part Still to be read and T the part that has been output Till so far. We use $+\!\!+$ as a notation for "followed by", and for simplicity we allow sequences as well as elements at both sides of $+\!\!+$. In words $pspec(p)$ says:

if initially $in = S$ and $out = \varnothing$ and $qspec(R, q)$ hold
then the call $p(q)$ establishes
the postcondition $out = sort(-\infty, R +\!\!+ S, +\infty)$ and $in = \varnothing$

and all this for *any* R, S and q. If $pspec(p)$ holds (or if it is assumed to hold as hypothesis), then we may conclude the correctness of

$\{in = S' \ \wedge \ out = \varnothing \ \wedge \ qspec(R', q')\}$
$p(q')$
$\{in = \varnothing \ \wedge \ out = sort(-\infty, R' +\!\!+ S', +\infty)\}$

for arbitrary expressions R', S' and q'; such a step in the correctness proof is called *instantiation* of $pspec(p)$ by $R, S, q := R', S', q'$. For instance, in the main program we may assert

$\{in = IN \ \wedge \ out = \varnothing \ \wedge \ qspec(\varnothing, q0)\}$
$p(q0)$ —instantiation of $pspec(p)$ by $R, S, q := \varnothing, IN, q0$
$\{in = \varnothing \ \wedge \ out = sort(-\infty, IN, +\infty)\}$

Now consider $qspec(R, q)$. Here T, m and n are to be chosen freely for each invocation of q, but R cannot because it isn't quantified in that formula: it is universally quantified in $pspec(p)$.

4. The Proof

We shall now present the proof of the correctness by annotating the program with assertions and specifications. But first we have to explain two notational conventions.

The first convention is this. Let S be a program fragment and P be an assertion so that S and P are interference free, that is the variables changed by S do not occur in P. Then we shall sometimes use the invariance of P over S, $\{P\}S\{P\}$, without further verification; (even the interference freeness will not be shown).

Secondly, let r be a procedure declared by, say, **proc** $r(x : int) = body$, let furthermore $(\forall\, x, y.\, \{P\}\, r(x)\, \{Q\})$ be a specification of r, and finally let t be a function mapping x, y into a well-founded ordering. To show the correctness of the specification by induction on t, one must show, for arbitrary x and y, $\{P\}body\{Q\}$, assuming as induction hypotheses all instantiations $\{P^{x\,y}_{a\,b}\}r(a)\{Q^{x\,y}_{a\,b}\}$ for which $t(a, b) < t(x, y)$. We shall annotate the program text as follows with the proof:

proc $r\,(x : int)$
: $\forall\, x, y.\, \{P\}r(x)\{Q\}$
= {{For arbitrary x, y (by induction on t): }}
$\{P\}\ body\ \{Q\}$.

We now present the annotated program, split into several parts.

- The main program:

$\{in = IN \ \wedge \ out = \varnothing\}$
proc $p\,(\textbf{proc}\ q\,(int, int)) : pspec(p) =$ 'see below';
proc $q0\,(m, n : int) : qspec(\varnothing, q0) =$ 'see below';
$\{in = IN \ \wedge \ out = \varnothing \ \wedge \ pspec(p) \ \wedge \ qspec(q0)\}$
begin $\{in = IN \ \wedge \ out = \varnothing \ \wedge \ qspec(q0)\}$
$p(q0)$ — instantiating $pspec(p)$ by $R, S, q := IN, \varnothing, q0$
$\{in = \varnothing \ \wedge \ out = sort(-\infty, IN, +\infty)\}$
end.

- The body of p:

{{For arbitrary R, S and q, by induction on $|S|$, the length of S: }}
$\{in = S \ \wedge \ out = T \ \wedge \ qspec(R, q)\}$
var v : int;
proc $q1\,(m, n : int)$: $qspec(R +\!\!+ v, q1)$ = 'see below';
$\{in = S \ \wedge \ out = T \ \wedge \ qspec(R, q) \ \wedge \ qspec(R +\!\!+ v, q1)\}$
begin $\{in = S \ \wedge \ out = \varnothing\}$
  if $in = \varnothing$
  then $\{in = \varnothing = S \ \wedge \ out = \varnothing\}$
    $q(-\infty, +\infty)$ —instantiating $qspec(R, q)$ by $T, m, n := \varnothing, -\infty, +\infty$
    $\{in = \varnothing \ \wedge \ out = \varnothing +\!\!+ sort(-\infty, R, +\infty)\}$
    $\{in = \varnothing \ \wedge \ out = sort(-\infty, R, +\infty)\}$
  else $\{in \neq \varnothing,$ say $in = i +\!\!+ S' = S, \wedge \ out = \varnothing \ \wedge \ qspec(R +\!\!+ v, q1)\}$
    $read(in, v)$;
    $\{in = S' \ \wedge \ out = \varnothing \ \wedge \ v = i \ \wedge \ qspec(R +\!\!+ v, q1)\}$
    $\{in = S' \ \wedge \ out = \varnothing \ \wedge \ qspec(R +\!\!+ i, q1)\}$
    $p(q1)$ —instantiating $pspec(p)$ by $R, S, q := R +\!\!+ i, S', q1$
    noting that $|S'| < |S|$
    $\{in = \varnothing \ \wedge \ out = sort(-\infty, (R +\!\!+ i) +\!\!+ S', +\infty)\}$
    $\{in = \varnothing \ \wedge \ out = sort(-\infty, R +\!\!+ S, +\infty)\}$
  fi
end
$\{in = \varnothing \ \wedge \ out = sort(-\infty, R +\!\!+ S, +\infty)\}$.

- The body of $q1$:

{{For arbitrary T, m and n (noting that $qspec(R, q)$ is assumed to hold true): }}
begin $\{out = T\}$
  if $m < v < n$
  then $\{out = T\}$
    $q(m, v)$; —instantiating $qspec(R, q)$ by $T, m, n := T, m, v$
    $\{out = T +\!\!+ sort(m, R, v)\}$
    $write(out, v)$;
    $\{out = T +\!\!+ sort(m, R, v) +\!\!+ v\}$
    $q(v, n)$; —instantiating $qspec(R, q)$
    by $T, m, n := T +\!\!+ sort(m, R, v) +\!\!+ v, m, v$
    $\{out = T +\!\!+ sort(m, R, v) +\!\!+ v +\!\!+ sort(v, R, n)\}$
    $\{out = T +\!\!+ sort(m, R +\!\!+ v, n)\}$ —because $m < v < n$
  else $\{out = T\}$
    $q(m, n)$ —instantiating $qspec(R, q)$ by $T, m, n := T, m, n$
    $\{out = T +\!\!+ sort(m, R, n)\}$
    $\{out = T +\!\!+ sort(m, R +\!\!+ v, n)\}$ —because not $m < v < n$
  fi
end
$\{out = T +\!\!+ sort(m, R +\!\!+ v, n)\}$.

- The body of $q0$:

$$\{\{\text{For arbitrary } T,\ m \text{ and } n:\}\}$$
$$\{out = T\}\ \mathbf{skip}\ \{out = T +\!\!+ sort(m, \varnothing, n)\}.$$

Once suitable $pspec(p)$ and $qspec(R, q)$ have been found and formulated, the correctness proof itself turns out to be straightforward. I claim that any valid informal explanation of the working of the program will closely parallel the above proof. It is also remarkable that the specification proved for $q1$ doesn't read $qspec(R +\!\!+ i, q1)$, but rather $qspec(R +\!\!+ v, q1)$. This is true even if $q1$ would have been declared *after* $read(in, v)$, as is possible in Algol68. It is only just before the *use* of $q1$ (as actual argument to p) that the former is derived from the latter. (And we could have made the transition from $R +\!\!+ v$ to $R +\!\!+ i$ even some lines later.)

Remark. It is well-known how to translate tail recursion into iteration. This is applicable to the above program. However, the resulting repetition can not be expressed as a Pascal program, and even not in Algol 68, because the global procedure variable q that is introduced to store the actual procedure arguments of p, has to contain procedures $q1$ that are formed locally in the repetition and therefor have a shorter extent (life time) than the procedure variable q, which is forbidden. Moreover, the 1-1 correspondence between a repetition and its tail recursive formulation also holds for the assertions and the annotation, so that this recursion removal brings hardly any simplification. Wiltink [10] performs this exercise.

5. The Same Algorithm in a Functional Language

It might be interesting to see the same algorithm expressed as a functional program, together with its correctness proof. The remarkable thing, now, is the much greater conciseness. We present the program in the style of Miranda [9].

$$\begin{array}{lll}
p\,(q, \varnothing) & = & q\,(-\infty, +\infty) \\
p\,(q, i +\!\!+ S') & = & p\,(q1, S') \\
 & & \quad \mathbf{where}\ q1\,(m, n) = q\,(m, i) +\!\!+ i +\!\!+ q\,(i, n)\,, \quad m < i < n \\
 & & \qquad\qquad\qquad\quad = q\,(m, n) \qquad\qquad\qquad\quad , \ \mathbf{otherwise} \\
q0\,(m, n) & = & \varnothing.
\end{array}$$

We now claim that the call $p(q0, IN)$ yields $sort(-\infty, IN, +\infty)$. To this end we shall prove the truth of $pspec'(p)$, defined as

$$\begin{array}{lll}
pspec'(p) & = & (\forall\, R, S, q.\ qspec'(R, q) \Rightarrow p(q, S) = sort(-\infty, R +\!\!+ S, +\infty)) \\
qspec'(R, q) & = & (\forall\, T, m, n.\ T +\!\!+ q(m, n) = T +\!\!+ sort(m, R, n))
\end{array}$$

Notice the strong similarity with $pspec(p)$ and $qspec(R, q)$; actually we may simplify the above formula by eliminating T. Again we prove $pspec'(p)$ by induction on $|S|$, the length of S:

Let R, S and q be arbitrary, and assume that $qspec'(R, q)$ holds.

case $S = \varnothing$:

$$\begin{array}{lll}
p(q, \varnothing) & = & q(-\infty, +\infty) \qquad\qquad \text{—use } qspec'(R, q) \text{ with } T, m, n := \varnothing, -\infty, +\infty \\
 & = & sort(-\infty, R, +\infty) \\
 & = & sort(-\infty, R +\!\!+ S, +\infty) \qquad\qquad\qquad\qquad \text{—because } S = \varnothing
\end{array}$$

case $S = i +\!\!+ S'$:

$$
\begin{array}{lll}
p(q, i +\!\!+ S') & = p(q1, S') & \\
& \textbf{where} & \\
& q1\,(m,n) & = q(m,i) +\!\!+ i +\!\!+ q(i,n)\,, \;\; m < i < n \\
& & = q(m,n) \qquad\qquad\quad , \;\; \textbf{otherwise} \\
& & \equiv \\
& & = sort(m,R,i) +\!\!+ i +\!\!+ sort(i,R,n)\,, \;\; m < i < n \\
& & = sort(m,R,n) \qquad\qquad\quad , \;\; \textbf{otherwise} \\
& & \equiv \\
& & = sort(m, R +\!\!+ i, n)\,, \;\; m < i < n \\
& & = sort(m, R +\!\!+ i, n)\,, \;\; \textbf{otherwise} \\
& & \equiv \\
& & = sort(m, R +\!\!+ i, n) \\
= sort(m, R +\!\!+ i +\!\!+ S', n) & \quad \text{—applying ind. hyp. with } R, S, q := R +\!\!+ i, S', q1 & \\
& \quad \text{—noting that } |S'| < |S| \text{ and that } qspec'(R +\!\!+ i, q1) \text{ holds} & \\
= sort(m, R +\!\!+ S, n) & \quad \text{—because } S = i +\!\!+ S' &
\end{array}
$$

So in both **case**s we have shown that $p(q, S) = sort(m, R +\!\!+ S, n)$ and thus $pspec'(p)$ has been proved. We conclude the proof of the claim by noticing that $q0\,(m,n) = sort(m, \varnothing, n)$ so that by instantiating $pspec'(p)$ with $R, S, q := \varnothing, IN, q0$ we get $p(q0, IN) = sort(-\infty, IN, +\infty)$.

Both the functional formulation of the "intricate" sorting algorithm and the proof of the functional program turn out to be quite conventional, whereas the imperative program is —for most people— rather unconventional.

Acknowledgement
I am grateful to Gerard Wiltink and Martin Rem for comments that have led to considerable improvement of the presentation.

References

1 Augusteijn, A. & Dijkstra, H., Sorting in Pascal without Structured Types. Document Nr. 207 - ESPRIT 415, (Philips Research Laboratories, Eindhoven, Netherlands), Nov 1986.
2 Eijk, P. van, A Useful Application of Formal Procedure Parameters. SIGPLAN Notices Vol. 21, 1986, Nr. 9, pp. 77-78.
3 Fokkinga, M.M., Structuur van Programmeertalen. Lecture Notes ("Structure of Programming Languages", in Dutch), University of Twente, Netherlands, 1983.
4 Hoare, C.A.R., An Axiomatic Basis for Computer Programming. Communications of the ACM, Vol. 12, 1969, Nr. 10, pp. 567-580, 583.
5 Kruseman Aretz, F.E.J., *lectures held in* Amsterdam, 1968.
6 Langmaack, H., On Correct Procedure Parameter Transmission in Higher Programming Languages. Acta Informatica Vol. 2, 1973, pp. 110-142.
7 Langmaack, H., On Procedures as Open Subroutines 1. Acta Informatica Vol. 2, 1973, pp. 311-333.
8 Langmaack, H., On Procedures as Open Subroutines 2. Acta Informatica Vol. 3, 1974, pp. 227-241.
9 Turner, D.A., Miranda - A non-strict functional language with polymorphic types. In: Functional Programming and Computer Architecture, (ed. J.-P. Jouannaud), LNCS Vol. 201, 1985, pp.1-16.
10 Wiltink, J.G., Another proof of Kuiper's Program. Manuscript JGW13, Technical University Eindhoven, Netherlands, Oct 1986.