Prive Werkexemplaar

ON THE CORRECTNESS PROOF OF THE SASL LAMBO PROGRAM

Maarten M Fokkinga

26 - 7 - 1981

revised 30 - 7 - 1981

Abstract We show that the proof given by Dijkstra for the imperative Lambo program is incomplete with regards to the correctness. Any completion of that proof, however, can be translated literally to a correctness proof for the corresponding SASL program. The techniques employed are quite generally applicable.

Contents

Section 0 Introduction

As a guest speaker at the summer school on Functional Programming (NewCastle upon Tyne, 20-31 July 1981), Dijkstra presented a GC program (GC abbreviates Guarded Command, see (Dijkstra 1976)) for which the corresponding SASL program (see (Turner 1979)) "could not easily be proved correct". The Lambo problem statement reads

> Let F be an unbounded ascending function on the natural numbers, define a function G on the natural numbers by
>
> $$G(y) = \text{the minimal } x \text{ with } F(x) > y .$$
>
> Now given F compute G .

Throughout this paper x, y and n vary over the natural numbers. The annotated GC program reads

$x, y := 0, 0 \; ; \; \{P_0 \wedge P_1\}$
do $F(x) = y \rightarrow x := x+1 \; \{P_0 \wedge P_1\}$
[] $F(x) > y \rightarrow g(y) := x \; ; \; y := y+1 \; \{P_0 \wedge P_1\}$
od

with the following invariant relations

P0: $F(x) \geq y \;\wedge\; \underline{A}\, i: 0 \leq i < x: F(i) \leq y$,

P1: $\underline{A}\, j: 0 \leq j < y: g(j) = G(j)$.

The SASL program reads

```
def g x y (p:q) =
    if p = y  →  g (x+1) y q
    [] p > y  →  x : g x (y+1) (p:q)
    fi;
def f x = F x : f (x+1);
g 0 0 (f 0) ?
```

Rather than assuming F available as a function Dijkstra chose to represent it by a typical SASL object: an infinite sequence. Similarly the result is a sequence rather than a function. (Note that the colon denotes appending an element to the head of a list; appending a list to a list is denoted by ++ . Priority of operators is as usual, but function application takes the highest priority; parentheses are optional.)

If I have understood it well, Dijkstra claimed that

(i) the invariance of P0 and P1 together with the nontermination of the GC program establish its correctness,

(ii) the SASL version cannot easily be proved correct, and

(iii) in particular, additional notation and/or terminology has to be introduced in order to express properties of sequences in the proof for the SASL program.

In the sections below we disprove each of these claims.

Section 1 The correctness of the GC program

The invariance of P_0 and P_1 are easily verified, as is the nontermination. However these cannot be considered as a correctness proof. For the following nonterminating repetition also maintains P_0 and P_1:

do true → skip **od** .

What we really want to assert about the program is that "eventually R holds", for some suitable relation R. For example, we might wish to assert that eventually g exceeds any value, say n. This choice of $\overset{R}{\text{}}$ is dealt with in section 3. Here and in section 2 we will take

$$R: \quad \underline{A}\, j: 0 \le j \le n: \quad g(j) = G(j)$$

where n is any fixed natural number. Another

equally valid choice for R is: $g(n) = G(n)$. This choice doesn't give any simplification in what follows in sections 1 and 2.

Proving the "eventuality" of R is like proving termination: we need to show that each step of the repetition makes some progress. The proper way of doing this is as follows:

> for that fixed value n occurring in R, give a variant function t (an integer valued expression in which e.g. n, x and y may occur), and show that from an invariant relation P,
> (i) each guarded command decreases t, and
> (ii) $t \geq 0 \vee R$.

It is this proof obligation which was not met for the GC program and which, as we will see, facilitates a direct translation to a correctness proof for the SASL program. Once t has been found, the verification of (i) and (ii) is easy. The reader might want to look for a suitable variant function himself.

Remark. The "eventuality" of R also implies that eventually R remains true forever. But R is not necessarily true ever since it becomes

true for the first time! (End of remark.)

In summary the correctness proof of the GC program entails to prove the following formulae.

(1) $P[x,y \leftarrow 0,0]$,

(2) $P \wedge B1 \Rightarrow P[x \leftarrow x+1] \wedge t[x+1+x] < t$,

$P \wedge B2 \Rightarrow P[y,g \leftarrow y+1, upd(g,y,x)] \wedge$
$t[y,g \leftarrow y+1, upd(g,y,x)] < t$,

(3) $P \Rightarrow B1 \vee B2$,

(4) $P \Rightarrow t \geq 0 \vee R$,

where $P = P0 \wedge P1$ and Bi $(i=1,2)$ are the guards in the program. In the sequel we assume these formulae to be true.

Section 2 The correctness of the SASL program

We consider it part of the challenge to deal with the sequence representations of functions in a direct, simple and clear way. In section 4 we briefly indicate how we could factorize the proof so that the sequence representation of F can be dealt with separately.

As a first step in the translation of the GC proof we reformulate relations P, R and the proof obliga-

tions (1)..(4) in order to deal with the sequence representation of the function called g in the GC program. It is obvious that the revised formulae (1')..(4') below easily follow from (1)..(4).

(1') $P'[x,y \leftarrow 0,0] \wedge lth\ [] = 0$,

(2') $P' \wedge B1 \wedge lth\ z = y$
$\Rightarrow P'[x \leftarrow x+1] \wedge t'[x \leftarrow x+1] < t' \wedge lth\ z = y$,
$P' \wedge B2 \wedge lth\ z = y$
$\Rightarrow P'[y,z \leftarrow y+1, z ++ x] \wedge t'[y,z \leftarrow y+1, z ++ x] \wedge lth(z ++ x) = y+1$

(3') $P' \Rightarrow B1 \vee B2$

(4') $P' \Rightarrow t' \geq 0 \vee R'$

where P', R' and t' are obtained from P, R and t by replacing the function symbol g by the sequence symbol z.

The next theorem states the correctness of the SASL program. We abbreviate $R'[z \leftarrow \ldots]$ by $R'(\ldots)$.

Theorem $R'(\ g\ 0\ 0\ (f\ 0)\ )$.

proof
It suffices to prove that for all x, y and z

(*) $P' \wedge lth\ z = y \Rightarrow R'(\ g\ z ++ g\ x\ y\ (f\ x)\ )$

because then instantiation with $x,y,z = 0,0,[]$ will do by virtue of (1'). So we prove (*); we

do it by induction on t' . We'll comment on the proof afterwards:

<u>case $t' < 0$</u>

From (4') we conclude R', i.e. $\underline{A}\, j : 0 \le j \le n : z_j = G_j$.
Hence $R'(z \mathbin{++} \text{"anything"})$ and in particular
$R'(z \mathbin{++} g\ x\ y\ (f\ x))$. QED

<u>case $t' \ge 0$</u>

We use some obvious notations to shorten the text of the proof.

$P' \wedge \mathit{lth}\ z = y$ {So by (3')}
$\overset{a}{\Rightarrow}$ $P' \wedge \mathit{lth}\ z = y \wedge (B1 \vee B2)$ {by sem of <u>if</u> and <u>or</u>}
$\overset{b}{=}$ if $B1 \to P' \wedge \mathit{lth}\ z = y \wedge B1$ {So by (2')}
$\overset{c}{\Rightarrow}$ $P'[x \leftarrow x+1] \wedge \mathit{lth}\ z = y \wedge t'[x \leftarrow x+1] < t'$ {by induction!}
$\overset{d}{\Rightarrow}$ $R'(z \mathbin{++} g\ (x+1)\ y\ (f\ (x+1)))$
$[]$ $B2 \to P' \wedge \mathit{lth}\ z = y \wedge B2$ {so by (2')}
$\overset{e}{\Rightarrow}$ $P'[\ldots] \wedge \mathit{lth}\ (z \mathbin{++} x) = y+1 \wedge t'[\ldots] < t'$ {by ind!}
$\overset{f}{\Rightarrow}$ $R'((z \mathbin{++} x) \mathbin{++} g\ x\ (y+1)\ (f\ x))$
$\overset{def}{=}$ $R'(z \mathbin{++} (x : g\ x\ (y+1)\ (F\ x : f\ (x+1))))$
fi

$\overset{b}{=}$ $R'(\text{if } B1 \to z \mathbin{++} E1 \ [] \ B2 \to z \mathbin{++} E2 \text{ fi})$
$\overset{d}{=}$ $R'(z \mathbin{++} \text{if } B1 \to E1 \ [] \ B2 \to E2 \text{ fi})$
$\overset{def}{=}$ $R'(z \mathbin{++} g\ x\ y\ (F\ x : f\ (x+1)))$
$\overset{e}{=}$ $R'(z \mathbin{++} g\ x\ y\ (f\ x))$. QED

<u>end of proof</u>.

The crucial steps are of course d and f . In step d the induction hypothesis (*) is applied for $x, y, z := x+1, y, z$; the condition $t'[x \leftarrow x+1] < t'$ guarantees that the new values of x, y and z are

closer to the basis than the old values. Similarly for step f. Steps h and l deal with the sequence representation of the function F; steps g and j deal with the sequence representation of the function called g in the GC program. The remaining steps occur also in the GC proof. To see this, notice that the GC proof actually reads as follows.

$$
\begin{aligned}
P &\overset{a}{\Rightarrow} P \wedge (B1 \vee B2) \\
&\overset{b}{=} \textbf{if } (i=1,2)\ Bi \to P \wedge Bi \ \textbf{fi} \\
&\overset{ce}{\Rightarrow} \textbf{if } (i=1,2)\ Bi \to P[---] \wedge t[---] < t \ \textbf{fi} \\
&\Rightarrow \textbf{if } (i=1,2)\ Bi \to wp(SLi, P \wedge t < to)[to \leftarrow t] \ \textbf{fi} \\
&= wp(IF, P \wedge t < to)[to \leftarrow t]
\end{aligned}
$$

Hence, together with $P \Rightarrow t \geq 0 \vee R$ {by (4)} we find {corresponding to steps d, f and k}

$$P \Rightarrow wep(DO, R)$$

by applying a theorem not proved here (and where wep stands for "weakest eventuality precondition"). Now using (1) we have

$$wep(x, y := 0, 0;\ DO,\ R)$$

It is needless to say that the variant function t plays an essential role. In both proofs it measures progress towards the goal. The omittance of mentioning it in the GC proof by Dijkstra has presumably been the reason to conjecture

that the SASL program is so difficult to prove. Indeed, as the equality in SASL means strong equality, and as G is a total function, assertions like

$$(z \;{+}{+}\; g\ x\ y\ (f\ x))\ j \;=\; G\ j$$

imply termination of execution of $(z \;{+}{+}\; g\ x\ y\ (f\ x))\ j$, and it is that very termination which isn't proved for the GC program by proving the invariance of P_0 and P_1 only.

By now the reader has presumably found a suitable variant function himself. If not, here is my choice:

t: $\quad G\,n - x + n - y$.

Thus progress is measured by the number of iteration steps, respectively recursive function calls, yet to be made before y exceeds n and consequently R becomes true forever.

Section 3 On the choice R: $y > n$

In view of P_1 the choice

R: $\quad y > n$

seems a suitable eventuality relation in order to conclude that the GC program is correct. However we feel that eventuality relations should not be formulated in terms of local variables, but only in terms of global variables. Clearly x and y are meant to be privars; in the SASL program they occur as parameters of g. It is therefore impossible to use R as an assertion about the sequence $g\ 0\ 0\ (f\ 0)$.

Nevertheless also the correctness proof for that choice of R is easily translated into a correctness proof for the SASL program, as shown below. We again use the truth of (1')..(4'); note that now R' is identical to R.

Theorem $P' \wedge lth\ z = y \Rightarrow (z +\!\!+ g\ x\ y\ (f\ x))\ n = G\ n$.

proof

By induction on t'.

case $t' < 0$

Then R' holds true, by (4'), so $lth\ z > n$,

so $(z +\!\!+ g\ x\ y\ (f\ x))\ n = z\ n = G\ n$ from P'. QED

case $t' \geq 0$

$(z +\!\!+ g\ x\ y\ (f\ x))\ n$

$= (z +\!\!+ \text{if } B1 \to E1 \ [] \ B2 \to E2 \text{ fi})\ n$

$= \text{if } B1 \to (z +\!\!+ E1)\ n \ [] \ B2 \to (z +\!\!+ E2)\ n \text{ fi}$

$= \text{if } B1 \to (z +\!\!+ g\ (x+1)\ y\ (f\ (x+1)))\ n$

$= \{$ because $P'[x \leftarrow x+1] \wedge t'[x \leftarrow x+1] < t'$

$\wedge\ lth\ z = y$ by (2'), we may apply

induction for x,y,z := x+1, y, z }
Gn

[] B2 → (z ++ x : g x (y+1) (f x)) n
= ((z ++ x) ++ g x (y+1) (f x)) n
= { because of (2') we may apply induction }
Gn

fi
= Gn . QED
end of proof

Now using (1') we find that for any n, g o o (f o) n = Gn
This expresses the correctness of the SASL program.

Section 4 Conclusion

It has been shown that even for nonterminating imperative programs variant functions play a crucial role in the correctness proofs, and that the variant functions may be used as the means on which to base an induction proof for the corresponding functional programs. (This is not to imply that such induction proofs are the only or the preferable way to prove the correctness; the functional style indeed allows for a variety of proof methods.)

Thus it turns out that the SASL lambo program is almost equally difficult or easy to prove correct as the GC program. There are however a few differences. First, in an informal reasoning it is for the GC program easy to separate concerns for termination (or eventuality) from concerns for partial correctness; this seems not as easy for the SASL program. On the other hand it is easy and natural to factorize the SASL proof in two parts; one part dealing with the correctness of

```
def g' x y =
    if Fx = y → g' (x+1) y ▯ Fx > y → x : g' x (y+1) fi
```

and the other part dealing with the representation of F by a sequence:

$$g'\ x\ y \;=\; g\ x\ y\ (f\ x) \quad .$$

(The proof of this equality is easy by computational induction; it suffices to show

$$\Omega\ x\ y \;=\; \Omega\ x\ y\ (f\ x)$$

and

$$g'\ x\ y = g\ x\ y\ (f\ x) \Rightarrow \text{"body } g'\text{"}(x,y) = \text{"body } g\text{"}(x,y,f\ x)) \quad .)$$

Finally we notice that from a formal point of view all what has been "proved" about the GC program needs a further justification by providing a definition for wep (weakest eventuality precondition) and a derivation of the theorem which justifies the proof method postulated in section 1 (cfr chapter 5 and 6 of (Dijkstra 1976)).

5 References

Dijkstra, E.W.: A discipline of programming. Prentice Hall, 1976.

Turner, D: SASL language manual. 1979.