TECHNISCHE HOGESCHOOL TWENTE

MEMORANDUM NR. 249

A SIMPLER CORRECTNESS PROOF OF AN IN-PLACE PERMUTATION ALGORITHM

MAARTEN M. FOKKINGA

MARCH 1979

Department of Applied Mathematics,
Twente University of Technology,
P.O. Box 217,
7500 AE Enschede, The Netherlands

ONDERAFDELING DER TOEGEPASTE WISKUNDE

Errata to A Simpler Correctness Proof of...

page	place	should read:
1	halfway	**do** $n \neq j \neq n-1 \rightarrow$
..	l.7 from below	... $j \leq n$ ...
2	l.2	... $j \leq n$ ..... **and** $\underline{A}\ i : 0..j-1 .\ f(i) = i$.
	l.8	$n \neq j \neq n-1 \rightarrow$ ...
	last line	... $j+1 \leq n$ ..... **and** $\underline{A}\ i : 0..j .\ f^*(i) = i$.
34	l.1 & l.14	.... $n \neq j \neq n-1$ ....

and, for the copies with a grey cover, also:

1	halfway	$q := F(j)$; **do** $q < j \rightarrow q := F(q)$ **od**;
		v: swap (j, q); $j := j+1$
4	line 10	$q := F(j)$; **do** $q < j \rightarrow q := F(q)$ **od** { $q = f(j)$ }
	12	.... **seq** $q, F(q), F^2(q), F^3(q)$ ...
	14	... do not contain q .
	16	... v : swap (j, q) ...

Contents

A SIMPLER CORRECTNESS PROOF OF AN IN-PLACE PERMUTATION ALGORITHM.

Abstract. In 1972 Duijvestijn gave a correctness proof of a particular permutation algorithm using an invariant relation. We present another proof based on this relation. It uses ghost variables and consequently can be split up into easily comprehensible parts. This might be of interest to the reader. The verification itself is hardly of interest: the application of the predicate transformation rules is straightforward and involves nearly no mathematics.

The program. The program to be proved correct rearranges a variable v: array [0 .. n-1] of T with initial value V , according to a given permutation F of 0 .. n-1 , using only auxiliary variables which do not depend on n or on T . Thus the program establishes

R: A i: 0 .. n-1 . v(i) = V(F(i)) .

Using here and in the sequel the notation of (Dijkstra 76), the program reads

```
j := 0;
do j ≠ n-1 →
    q1 := F(j); do q1<j → q1 := F(q1) od;
    v :swap(j,q1); j := j+1
od .
```

We will give the correctness proof together with a construction of the program. We want to stress again that the verfication of the invariant relations is merely a boring formula manipulation, involving no interesting mathematics. We present it only to contrast it with (Duijvestijn 72).

A "stepping stone" program: using a ghost variable

We try to establish R by means of a repetition with the invariant relation (found by standard techniques)

0 ≤ j ≤ n and A i: 0 .. j-1. v(i) = V(F(i)) .

In order to know how the remaining elements of v need to be arranged yet, we introduce an array variable f , representing a permutation of j .. n-1 , such that

A i: j .. n-1. v(f(i)) = V(F(i)) .

Letting f(i) = i for i: 0 .. j-1 , we can express the complete invariant relation as P0 and P1 :

P0: f denotes a permutation of 0 .. n-1 ,

P1: $0 \le j \le n$ and A i: 0 .. n-1. v(f(i)) = V(F(i)) and A i: ~~j .. n-1~~ 0 .. j-1. f(i) = i .

The program, then, has the structure

f := F; j := 0 {v = V; hence P0 and P1 established};

do "maintain P0 and P1, decrease n-j" od {R} .

There are several ways to derive or invent the refinement

"maintain P0 and P1, decrease n-j":

~~n≠j≠n-1~~ → v :swap(j,f(j));f :swap(f^{-1}(j),j); j := j+1

~~In general n≠j has been strengthened for reason of efficiency~~ If you wish, you may replace the general by j<n-1.

We will give the verification of the invariance only.

Recall the semantics of assignment and swapping.

wp(x := e, P) = P[x ← e] .

wp(a :swap(x,y), P) = P[a ← a'], where a' = a[x ← a(y), y ← a(x)] .

In general, the array value a' = a[x ← e1, y ← e2] is defined

for any a, x, y, e1, e2 with $x \neq y$ or e1=e2 , as follows

$$a'(i) = \begin{cases} a(i) & \text{for i different from x and y} \\ e1 & \text{for i = x} \\ e2 & \text{for i = y .} \end{cases}$$

Now we prove the invariance; first of P0 and then of P1.

Because f is subject to swap only, P0 is kept invariant. Formally this is shown as follows.

wp(v :swap(j, f(j)); f :swap(f^{-1}(j),j); j := j+1, P0) =

= ((P0[j ← j+1])[f ← f'])[v ← v']

where f' = f[f^{-1}(j) ← f(j), j ← f(f^{-1}(j))], v' = ...

= f' is a permutation of 0 .. n-1

= f ∘ p is a permuation, where p is the pair exchange "f^{-1}(j) <--> j"

and this holds true, because f being a permutation on account of P0,

and p being a permutation, so is the composition f ∘ p .

(Note that, f being a permutation, the inverse f^{-1} is well defined!)

wp(v :swap(j, f(j)); f':swap(f^{-1}(j), j); j := j+1, P1) =

= ((P1[j ← j+1])[f ← f'])[v ← v']

where f' = f[f^{-1}(j) ← f(j), j ← f(f^{-1}(j))]

v' = v [j ← v(f(j)), f(j) ← v(j)]

= $0 \le j+1 \le n$ and A i: 0 .. n-1. v'(f'(i)) = V(F(i)) and A i: ~~0 .. j~~ ~~j+1 .. n-1~~ f'(i) = i.

The first term is implied by P1 and $j \neq n-1$; we prove

a: $v'(f'(i)) = V(F(i))$, and

b: $i>j$ or $f'(i) = i$

from P1 by cases on i:

For $f^{-1}(j) \neq i \neq j$:

a. $v'(f'(i)) =$ (def f':) $v'(f(i)) =$ (def v':) $v(f(i)) =$ (from P1:) $V(F(i))$,

b. $f'(i) =$ (def f':) $f(i) =$ (from P1:) i, if $i \leq j$.

For $i = j$:

a. $v'(f'(j)) =$ (def f':) $v'(j) =$ (def v':) $v(f(j)) =$ (from P1:) $V(F(j))$,

b. $f'(j) =$ (def f') j .

For $i = f^{-1}(j)$:

a. $v'(f'(i) =$ (def f':) $v'(f(j)) =$ (def v':) $v(j) = v(f(f^{-1}(j)) =$ (from P1:) $V(F(i))$,

b. (from P1:) A i: 0 .. j-1. $f(j) = i$, hence $i = f^{-1}(j) \geq j$. Now either $i = f^{-1}(j) > j$, or $i = f^{-1}(j) = j$ and $f'(i) = f'(j) = j = i$.

Final program: the ghost variable eliminated

There is an additional invariant relation, which enables us to eliminate variable f :

P2: A i: j .. n-1. $f(i)$ = first elt in the seq $F(i), F^2(i), F^3(i)$ ... which is $\geq j$.

Here follows the proof of the invariance of P2.

wp(v :swap(j, f(j)); f :swap($f^{-1}(j)$, j); j := j+1, P2) =

= ((P2[j ← j+1])[f ← f'])[v ← v']

where $f' = f[f^{-1}(j) \leftarrow f(j),\ j \leftarrow f(f^{-1}(j))]$

and $v' = v[j \leftarrow v(f(j)),\ f(j) \leftarrow v(j)]$

= A i: j+1 .. n-1. $f'(i)$ = first elt in the seq $F(i), F^2(i), F^3(i)$ ... which is $\geq j+1$.

We prove the requirement for $f'(i)$ from P2 by cases on i.

For $j+1 \leq i \leq n-1$ and $i \neq f^{-1}(j)$:

$f'(i) =$ (def f':) $f(i)$ {and this is > j on account of P0 and P1}

$=$ (from P2) the first elt in $F(i), F^2(i)$ ... which is $\geq j$,

so $f'(i)$ = the first elt in the seq $F(i), F^2(i)$ ... which is $\geq j+1$.

For $j+1 \leq i \leq n-1$ and $i = f^{-1}(j)$:

f'(i) = (def f':) f(j) {and this is > j on account of P0, P1 and $f^{-1}(j) \neq j$}

= (from P2:) the first elt in F(j), $F^2(j)$ ... which is ≥ j ,

so f'(i) = the first elt in the seq F(j), $F^2(j)$ ... which is ≥ j+1 ... (*)

Also j = $f(f^{-1}(j))$ = f(i)

= (from P2:) the first elt in F(i), $F^2(i)$ ... which is ≥ j ... (**)

Combining (*) and (**) yields

f'(i) = the first elt in the seq F(i), $F^2(i)$ ... which is ≥ j+1 .

This, by the way, is the most non-trivial step of all verifications.

Hence, just before v :swap(j, f(j)) we may compute f(j) as follows:

q := F(j); do q < j → q := F(q) od {q = f(j)} .

The invariant relation of the repetition reads:

f(j) = the first elt in the seq q, F(q), $F^2(q)$, $F^3(q)$ ... which is ≥ j.

The verification is easy, and is left to the reader. In addition P0, P1, P2, and j≠n are invariant as well, because they do not contain q .

Once the above line has been inserted, and v :swap(j, f(j)) has been replaced by v :swap(j, q) , it appears that f is not used at all -- except in updatings of itself -- and may therefore be deleted. So we have proved the correctness of the given program.

In conclusion. The ghost variable f has enabled us to split the program construction and the invariant relation in two easily comprehensible and separately verifiable parts. The preliminary mathematical properties proved by (Duijvestijn 72) have, more or less, been verfied during the straightforward and, indeed, rather boring verification of the invariants. Thus the only interesting feature of the correctness proof is the formulation of an elegant invariant.

References

Dijkstra, E.W.: A Discipline of Programming, Prentice Hall (1976).

Duijvestijn, A.J.W.: Correctness proof of an in-place permutation, BIT 12 (1972) 318-324.

Acknowledgement. I thank Doaitse Swierstra for pointing out some sloppiness in a previous version of this paper.