The Bounded Linear Search

The problem of the Bounded Linear Search can be specified as follows. Given a boolean array (or function) $b(i: 0 \leq i \wedge i < N)$, $0 \leq N$, establish

$$[\![\underline{\text{var}}\ x : \text{int} : \{Q\}\ \text{Bounded Linear Search}\ \{R\}]\!],$$

in which Q and R are given by

Q: true

R: $R0 \wedge R1 \wedge R2$.

where

$R0$: $0 \leq x \wedge x \leq N$

$R1$: $(\underline{A}i: 0 \leq i \wedge i < x: \neg b.i)$

$R2$: $x = N \vee b.x$.

The above is a formalization of the problem's "folk-specification", which may run as "set x to the first occurrence of value true in array b, if any, and set it to N otherwise".

* * *

The attentive reader may very well feel uneasy at our specification since the second disjunct of $R2$ is not defined for $x = N$. This uneasiness is fully justified: it is the occurrence of the

"undefined" b.N which turns the problem of the Bounded Linear Search into one that is a little nastier than the regular Linear Search. Therefore, we examine the specification a bit more carefully.

The specification grants us a function b which is "defined" on a limited domain. We view this "definedness" as "computable" inside the domain and as "non-computable" outside the domain. As a consequence, we have to see to it that a program at work will never evaluate a function outside the specified domain. But outside the realm of computation, nothing prevents us from thinking of b as a total function, having a value in any point we like, viz. the prescribed value inside the specified domain and an irrelevant (or benevolent) value outside that domain. Thus, the troublesome expression R2 is no longer suspect, as long as we see to it that it is never evaluated for x outside the domain $0 \leq x \wedge x < N$.

The latter remark discards an immediate solution, emerging from a rather direct application of the regular Linear Search:

```
x := 0   { inv. R0 ∧ R1 }
; do ¬ (x = N ∨ b.x)        {i.e. ¬ R2}
→   x := x + 1
od .
```

This is one of the most well-known programs on earth; it derives its reputation from the associated error-message: "index-out-of-range". (We wish to add, however, that the program does return a correct result when run on a machine that in one way or another -justifiably or not- can produce the irrelevant value b.N .)

* * *

Now, let us embark on a derivation for the Bounded Linear Search. First we carry out a very small cosmetic change towards simplifying expression R2. We introduce thought element b.N with value true, so that we then have $x = N \Rightarrow b.x$. This enables us to rewrite Q and R2 from the original specification into

Q: b.N

R2: b.x .

Next, we try to proceed as with the derivation of the Linear Search, dividing the various conjuncts of postcondition R over the invariant and the guard. As before, R0 and R1 will have to be part of the invariant. They require initialization $x := 0$. But this time expression R2 is not allowed to occur

in the guard, because b.x is not computable for x = N – a value for x that we cannot exclude! – . Therefore, postcondition R2 has to follow from the invariant as well. Adding R2 to the invariant too, however, will require x := N as an initialization, which raises a conflict with the established initialization x := 0 . The dilemma we now run into is a standard one, and so is the way out.

The impossibility to initialize the designated invariant indicates a lack of at least one degree of manipulative freedom. The freedom is obtained by parametrization, which is a technique of generalization often referred to as "replacing a constant by a variable". Although, in general, there are numerous possibilities for parametrization, the developed scenario for our current problem offers only very little choice.

In order to remove the initialization conflict, we introduce a fresh variable y, and the initialization will become

x, y := 0, N .

From precondition Q we conclude that y := N establishes relation

P2: b.y .

which will be added to the invariant instead of R_2. Now, postcondition R_2 follows from P_2 if in addition we have $y = x$; and this tells us that the guard should be $y \neq x$.

Summarizing all the above, we have derived the following program as a stepping stone towards the Bounded Linear Search :

$\{Q\}$
$x, y := 0, N$
$\{\text{inv. } R0 \wedge R1 \wedge P2\}$
; $\underline{\text{do}}$ $y \neq x \rightarrow \ldots$ $\underline{\text{od}}$
$\{R0 \wedge R1 \wedge P2 \wedge y = x, \text{ hence } R\}$.

where

Q: $b.N$
$R0$: $0 \leq x \wedge x \leq N$
$R1$: $(\underline{A}\, i : 0 \leq i \wedge i < x : \neg\, b.i)$
$P2$: $b.y$
$R2$: $b.x$

* * *

In the step of the repetitive construct the elements of b must play a rôle. The two expressions $b.x$ and $b.y$ present themselves as candidates for being mentioned. But whatever choice we make we have to ensure that they do not refer to a non-computable element

of b. In view of y's initial value, a reference to b.y is deadly dangerous, since b.N is non-computable. Therefore we focus on b.x. By R0 we already have $0 \leq x \wedge x \leq N$ as a precondition of the step. Can guard $y \neq x$ assist in concluding $x \neq N$? It does if we adopt as an additional invariant

P3: $x \leq y \wedge y \leq N$.

which -indeed- is correctly established by the given initialization.

Now we are through. The remaining part of the design is driven by the observation that -as in the regular Linear Search- $x := x+1$ is the only thinkable modification of x. The rest follows. For the Bounded Linear Search we derived (the unannotated text)

```
x, y := 0, N
; do y ≠ x →
     if ¬b.x → x := x+1
     [] b.x → y := x
     fi
  od
```

The reader may verify that y-x is a suitable variant function. Its boundedness from below follows from P3.

x x x

The (Bounded) Linear Search belongs to the most fundamental algorithms in computing. Also, it is one of the most widely applied ones. Isn't it amazing then, that after decades of experience in computing, Linear Searches still form such a trouble spot in the everyday practice of programming? No, it isn't. The only way that I know of to come to grips with an algorithm – even if it is a simple one – is by means of its formal specification and its formal derivation, and – as everyone knows – this is not typical of the way in which professional programmers perform.

With this sad conclusion, nothing else remains to be done than to recommend to the reader who cares, that he now grab a piece of paper no larger than an envelope and use it to summarize the (technical) heart of the foregoing development of the Bounded Linear Search.

April & June 1990

W.H.J. Feijen
Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O. Box 513
Eindhoven
The Netherlands.