Copyright Notice

The manuscript was published as

Structured Programming 10 (1989), 1: 5–9.

The linear search revisited

by Edsger W. Dijkstra[0] and W.H.J. Feijen[1]

0) Department of Computer Sciences, The University of Texas at Austin, Austin TX 78712-1188, USA

1) Department of Mathematics and Computing Science, Technological University Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands.

This paper is about some aspects of the formal derivation of imperative programs. In the usual setting, we start from a given functional specification and have to design a program meeting this specification. This setting is analogous to being given a conjecture and having to turn it into a theorem by designing a proof for it. But mathematical development is often less unidirectional: while designing the proof, we construct the theorem proved by it. In our first program derivation, we shall proceed in the latter vein: the specification will be constructed in a number of steps, as dictated by the desire to proceed with the design of our program. We have chosen this form of presentation in order to stress which parts of the specification are responsible for which design decisions. We shall illustrate our approach with some of the simplest examples

possible. Here we go!

We want to design a program manipulating an integer variable n so as to establish a postcondition R. As long as nothing is known about R, there is nothing we can do. In order to make the problem minimally more specific, let us postulate that, viewed as equation in n, R has a unique solution – in other words, that we are heading for a semantically deterministic program –. This choice says little about the program, but something about R: more precisely, all it says about R is that there exists an integer N such that

(0) $$[R \equiv n = N] \quad .$$

Our task has been pinned down to the design of a program establishing the postcondition $n = N$.

The task of writing a program that establishes $n = N$ depends on how N has been defined. For instance,

Approximation 0: $n := N$

would do if the value of N were given directly. For the sake of argument we explore the situation in which Approximation 0 has to be discarded because N is given indirectly by a number of properties.

Now it is time to remember that there is another way of establishing $n = N$, viz. with a program ending with

$$\underline{\text{do}}\ n \neq N \rightarrow \ldots\ldots\ \underline{\text{od}}$$

but for that to work, we need for the repetition an invariant and a termination argument. Let it be given that N is natural, i.e.

(1) $\quad 0 \leq N$.

Which invariants can we initialize thanks to this fact? Expression (1) contains two constants that can be replaced by our variable n. Of the two Hoare triples

$$\{0 \leq N\}\ n := N\ \{0 \leq n\} \quad \text{and}$$

$$\{0 \leq N\}\ n := 0\ \{n \leq N\}\ ,$$

the first one is not interesting for the initialization of n before the repetition — the repetition would boil down to a skip and the whole solution to the discarded Approximation 0 —. So we take the second Hoare triple to suggest initialization and invariant; thus we arrive at the annotated

Approximation 1:

$$\begin{array}{l} \{0 \leq N\}\ n := 0\ \{n \leq N\} \\ ;\ \underline{\text{do}}\ n \neq N \rightarrow \{n \leq N \wedge n \neq N\}\ \ldots\ldots\ \{n \leq N\}\ \underline{\text{od}} \\ \{n = N\} \end{array}$$

We still have to supply the termination argument, which -among other things- requires that the as yet unspecified statement "....." differ from skip. After simplification of its precondition $n \leq N \wedge n \neq N$ to $n+1 \leq N$, we see that $n := n+1$ substituted for "....." maintains the invariant $n \leq N$, which implies that $N-n$ is bounded from below. Since $N-n$ is decreasing with respect to $n := n+1$, this completes the termination argument. Incorporating the observation that we can strengthen the invariant with the conjunct $0 \leq n$, we arrive at

Approximation 2:

$$\{0 \leq N\}\ n := 0\ \{0 \leq n \wedge n \leq N\}$$
$$;\ \underline{\text{do}}\ n \neq N \rightarrow n := n+1\ \{0 \leq n \wedge n \leq N\}\ \underline{\text{od}}$$
$$\{n = N\}\ .$$

Approximation 2 is all we can do if (1) is all that has been given about N.

How can we eliminate N from the guard? Well, there are two things to know. Firstly, that our beloved inference rule, the principle of Leibniz,

$$x = y \Rightarrow f.x = f.y$$

can also be written as

$$x \neq y \Leftarrow f.x \neq f.y\ ,$$

and, secondly, that termination and invariance proofs remain valid under strengthening of the guards of a repetition. Let b be a boolean function on the integers, and let it be given that N satisfies

(2) b.N .

Then we observe

$$
\begin{array}{ll}
 & n \neq N \\
\Leftarrow & \quad \{\text{Leibniz}\} \\
 & b.n \neq b.N \\
= & \quad \{(2)\} \\
 & \neg\, b.n
\end{array}
\quad ,
$$

which leads to

Approximation 3:

$$
\begin{array}{l}
n := 0 \\
\mathbf{;}\ \underline{\mathbf{do}}\ \neg\, b.n \rightarrow n := n+1\ \underline{\mathbf{od}} \\
\{0 \leq n \wedge n \leq N \wedge b.n\}
\end{array}
\quad .
$$

This is very nice, because now N no longer occurs in the program text. We have written the guaranteed post condition - viz. the conjunction of the invariant and the negated guard - in full because, by strengthening the guard we have weakened the post condition. Our final obligation is to show how to derive

R, i.e. $n=N$, from the guaranteed postcondition. To this end we observe

$$
\begin{array}{ll}
 & 0 \leq n \wedge n \leq N \wedge b.n \Rightarrow n = N \\
= & \quad \{\text{predicate calculus}\} \\
 & 0 \leq n \wedge n \leq N \wedge n \neq N \Rightarrow \neg b.n \\
= & \quad \{\text{arithmetic}\} \\
 & 0 \leq n \wedge n < N \Rightarrow \neg b.n \\
= & \quad \{\text{see (3), below}\} \\
 & \text{true}
\end{array}
$$

where N is postulated to satisfy

(3) $\quad (\underline{A}\, x: 0 \leq x \wedge x < N: \neg b.x)$.

We are done with our design: Approximation 3 is the program that meets the specification consisting of the conjunction of (1), (2), and (3), which defines N as the <u>smallest</u> <u>natural</u> number for which b holds. We are all well-acquainted with this program; it is known as The Linear Search. A few remarks are in order.

Firstly, our exploration of what could be done on account of (1) alone is no joke. We recently saw a proof of a theorem about prime numbers being simplified; the first (and crucial) step of the simplification consisted in exploiting the simplest property of prime numbers, viz. that they are positive.

Secondly, the "invention" of "n:=n+1" for "....." was in a sense forced: with a formal specification

$$\{P^n_E\} \ldots\ldots \{P\}$$

the Axiom of Assignment tells us that n:=E is "the solution".

Thirdly, the strengthening of the invariant with $0 \leq n$ could have been done more constructively. The initialization would have justified the conjunct $0 = n$; the requirement of invariance under (repeated execution of) n:=n+1 then dictates the weakening of $0 = n$ to $0 \leq n$.

Fourthly, in programs like Approximation 2 there is always the question whether to choose $n \neq N$ or $n < N$ as the guard. For not very good reasons we had adopted $n \neq N$. The above development gives a reason: since we can always strengthen the guard of a repetition later, we should not do so prematurely.

Fifthly, we would like to point out that the quantified expression (3) enters the picture only at the very end, and is nowhere manipulated in the derivation. (Compare [0] and [1].)

Sixthly, solutions to the Saddleback Search and the Welfare Crook (see [1]) and similar problems are most easily designed along the

above lines.

* * *

We now turn to a seemingly very similar problem that on closer inspection turns out to be a little bit nasty. It is known as The Bounded Linear Search. For $D \geq 0$ and d a boolean function on the integers, we are invited to design a program such that n eventually equals the smallest natural value $< D$ such that $d.n$, if such a value exists, and that otherwise ends with $n = D$.

To this end — and here the reader may recognize the "sentinel" — we apply The Linear Search to the boolean function b, given by

(4) $$b.x \equiv x = D \vee d.x$$

Since from (4) we derive $b.D$, we conclude that this application of The Linear Search terminates. Substituting according to (4) in The Linear Search we get

```
n := 0
; do ¬ (n = D ∨ d.n) → n := n+1 od
  {R}
```

where we derive for the postcondition R

$$
\begin{array}{ll}
 & R \\
= & \{(1),(2),(3) \text{ with } (4) \text{ and } N:=n\} \\
 & 0 \leq n \wedge (n = D \vee d.n) \wedge \\
 & (\underline{A}\, x:\ 0 \leq x \wedge x < n:\ x \neq D \wedge \neg d.x) \\
= & \{\text{predicate calculus}\} \\
 & 0 \leq n \wedge (\underline{A}\, x:\ 0 \leq x \wedge x < n:\ x \neq D) \wedge \\
 & (n = D \vee d.n) \wedge \\
 & (\underline{A}\, x:\ 0 \leq x \wedge x < n:\ \neg d.x) \\
= & \{\ 0 \leq D\ ;\ (5), \text{ see below}\} \\
 & 0 \leq n \wedge n \leq D \wedge (n = D \vee d.n) \wedge F.n \\
= & \{\text{predicate calculus}\} \\
 & (n = D \wedge F.n) \vee (0 \leq n \wedge n < D \wedge d.n \wedge F.n) \\
= & \{\ (6),\ (7),\ \text{see below}\} \\
 & R0 \vee R1
\end{array}
$$

with the definitions for F, $R0$, and $R1$:

(5) $[F.n \equiv (\underline{A}\, x:\ 0 \leq x \wedge x < n:\ \neg d.x)]$

(6) $[R0 \equiv n = D \wedge F.n]$

(7) $[R1 \equiv 0 \leq n \wedge n < D \wedge d.n \wedge F.n]$.

The above disjunctive form for R fully confirms that -as expected- the program does the job. But it suffers from a major shortcoming.

Execution of the linear search as in Approximation 3 never evaluates values of b that are irrelevant for the definition of N, viz b.x for $x < 0 \vee x > N$. In our last

program, however, evaluation of d.D is not excluded even though the value of d.D is irrelevant for the determination of the final state. It is here that we encounter a difference between traditional mathematics and computing. In traditional mathematics, there is nothing wrong with manipulating eventually irrelevant values — for instance, prior to their elimination — . In computing, however, the price of evaluating an irrelevant value can be heavy (in fact, can be viewed as unbounded if the environment does not provide an effective algorithm for computing such irrelevant values). As it stands, our last program therefore has to be rejected for computational reasons.

It has been suggested that such programs be saved by redefining the logical connectives

$p \vee q \equiv$ (if p then true else q)
$p \wedge q \equiv$ (if p then q else false) , etc.

(To distinguish these "conditional connectives" from the genuine ones, they have been denoted by "cor", "cand", etc.) But the price to be paid for such redefinitions is heavy: we go from two-valued to three-valued logic, disjunction and conjunction are no longer symmetric and no longer distribute over each other, etc. Manipulation of these quasi-boolean expressions becomes in fact so complicated that we must

consider the redefinition of the logical connectives a strategic mistake. So much for the efforts to save our last program by redefinition of the connectives.

So now we set ourselves the task of designing a program that establishes $R0 \vee R1$ without calling for the evaluation of irrelevant values. Because a disjunction in the postcondition can, in general, be dealt with by providing alternatives, we allow ourselves for a start to consider a partial program for the establishment of $R0$, i.e. partial in the sense that it establishes $R0$ if possible, and otherwise fails to terminate. We propose that it establish $n = D$ to begin with and subsequently fail to terminate in the case $\neg F.n$. The relevant properties of F -see (5)- are

$$[F.0] \text{ and } [F.m \wedge \neg d.m \equiv F.(m+1)] \text{ for } 0 \leq m .$$

Partial Program:

$n := D$
; $[\![$ var m: int ; $m := 0$ $\{n = D \wedge 0 \leq m \wedge m \leq n \wedge F.m\}$
; do $m \neq n \rightarrow$ $\{n = D \wedge 0 \leq m \wedge m < n \wedge F.m\}$
if $\neg d.m \rightarrow$ $\{n = D \wedge 0 \leq m \wedge m < n \wedge F.m \wedge \neg d.m\}$
$m := m+1$ $\{n = D \wedge 0 \leq m \wedge m \leq n \wedge F.m\}$
fi $\{n = D \wedge 0 \leq m \wedge m \leq n \wedge F.m\}$
od $\{n = D \wedge F.n\}$
$]\!]$ $\{R0\}$.

The program is partial because, as it stands, the if..fi construct aborts in the case d.m . In order to cater for that case, we provide it with a second guarded command, beginning with

$$d.m \rightarrow \{n=D \wedge 0 \leq m \wedge m<D \wedge F.m \wedge d.m\} \ldots\ldots$$

Note that the last 4 conjuncts of the above assertion are $R1$ with n replaced by m ! So, for the second guarded command we suggest

$$d.m \rightarrow n := m \ \{R1 \wedge m=n\} \quad ,$$

at the same time weakening the invariant with the disjunct $(R1 \wedge m=n)$. Note that this weaker invariant suffices for the same precondition for the if..fi construct. Thus we arrive at our final program with invariant P

$$[P \equiv (n=D \wedge 0 \leq m \wedge m \leq n \wedge F.m) \vee (R1 \wedge m=n)]$$

and variant function $n-m$

```
  n:= D
; |[ var m: int; m:= 0 {P}
  ; do m ≠ n →
         if ¬ d.m → m:= m+1 [] d.m → n:= m fi {P}
    od {P ∧ m=n, hence}
  ]| {R0 ∨ R1}
```

Note that in our effort to make the partial program total, we could have followed a different line of reasoning. If we had decided to replace abortion by termination of the repetition,

$$d.m \rightarrow n := m \quad \text{and} \quad d.m \rightarrow m := n$$

would be the simplest options for the second guarded command. The choice of the former would have generated the disjunct $R1$ of the postcondition of our final program for The Bounded Linear Search.

A final question to our readers: did you know this program for The Bounded Linear Search? We did not. That our exercise in program derivation led to a new solution for such an old problem was a very encouraging experience.

Acknowledgement We are indebted to the Austin Tuesday Afternoon Club, in particular to Ken Calvert, who suggested for the guard in the Partial Program $m \neq n$ instead of our original $m \neq D$.

[0] Dijkstra, Edsger W., "A Discipline of Programming", Englewood Cliffs, New Jersey: Prentice-Hall, 1976.

[1] Gries, David, "The Science of Programming", New York, Springer-Verlag, 1981.