prive p.4

A NOTE ON THE PRAGMATICS OF THE REPETITIVE CONSTRUCT

by

Maarten M. Fokkinga
Twente University of Technology
Department of Applied Mathematics
P.O. Box 217, 7500 AE ENSCHEDE
The Netherlands

October 1978

Abstract. We formulate and discuss three pragmatics, i.e. thumb rules for the use, of the repetitive construct, and illustrate them by means of examples taken from "A Discipline of programming" (Dijkstra 76).

Keywords and phrases. Guarded commands, repetitive construct, while statement, recursive refinement.

Contents.

1. Introduction and conclusion

In this paper we distinguish several pragmatics (that is, thumb rules for the use) of the repetitive construct of (Dijkstra 76). One of them is the traditional pragmatics of the while statement. We have objections to it and consider it old fashioned; but in spite of this, it is used several times in (Dijkstra 76). Our formulation of the second pragmatics almost literally occurs in (Dijkstra 76), and of course is used there a lot of times. A third pragmatics is a combination of the other two. We show for some very simple examples the results of the various pragmatics, and conclude with the observation that, unfortunately, in some cases the repetitive construct is not suitable to express both the conceptual algorithm as clear as possible and at the same time an executable, although possibly inefficient, program for the algorithm.

2. The Pragmatics

Each of the pragmatics gives us a strategy to follow in order to establish a given relation R by means of a repetition. The strategy doesn't guarantee success; if not succesful we should revise the development of the algorithm.

In the sequel we use the following abbreviations: est: establish, gvn: given, mnt: maintain, dcr: decrease.

The first pragmatics seems to be used widely for the while statement.

Pr1: First invent relations P and Q such that P <u>and</u> Q $\Longrightarrow$ R , and invent a variant function T (such that P $\Longrightarrow$ T $\geq$ 0). Then try to refine the scheme "est P"; <u>do</u> <u>not</u> Q $\rightarrow$ "gvn <u>not</u> Q mnt P dcr T" <u>od</u>.

Sometimes the attempt to refine the scheme fails because its command doesn't necessarily decrease T , whereas it would do so if the guard <u>not</u> Q is strengthened into <u>not</u> Q' . Then the altered scheme only establishes P <u>and</u> Q' and some further statements are requested to establish R from P <u>and</u> <u>not</u> Q' .

Other reasons of failure to refine the scheme might be remedied by inventing another T or even new P and Q .

Pragmatics Pr1 seems to be practised frequently in (Dijkstra 76). In particular this is clear from the explanatory text and the program schemes in the following examples in chapter 8, "the formal treatment of some small examples":

E1: second example, p. 53; we quote:

```
do j≠n → a step towards j=n under invariance of P od,
do j≠n → if f(k)≥f(j) → j:=j+1
           ▯ f(k)≤f(j) → k,j:=j,j+1 fi od .
```

E2: second version of fifth example, p. 62; we quote:

```
do a+1≠b → decrease b-a under invariance of P od.
```

E3: sixth example, p. 65; wequote:

```
do h≠1 → squeeze h under invariance of P od,
do y≠0 → do 2/y → x,y:=x*x,y/2 od;
         y,z:=y-1,z*x
od .
```

E4: seventh example, p. 67; we quote:

```
do j≠n → allsix:=allsix and f(j)=6; j:=j+1 od.
```

E5: eight example, p. 70; we quote:

```
do s≠r → {...} increase s by a suitable amount under
         invariance of P1 {...}
od .
```

We seriously object to pragmatics Pr1 . The basic idea of guarded commands is that a command is guarded by (a condition implying) its wp(-,-) and wdec(-,-). In Pr1 and each of the above program schemes, however, the guard not Q arises for quite other reasons; it is in fact already chosen before the command itself is known!! In addition, Pr1 only leads to repetitive constructs with a single guarded command (, that is, the old fashioned while statement!).

The second pragmatics does justice to the term "guarded command".

Pr2: First invent a relation P such that for some Q we have P and Q ⟹ R , and invent a variant function T (such that P ⟹ T≥0). Then try to refine the scheme "est P"; do "mnt P dcr T" od .

Thus we should invent some statement lists SLi , and derive conditions Bi such that for each i

P and Bi $\Longrightarrow$ wp(SLi,P) and wdec(SLi,T) .

The repetition then reads do B1 $\rightarrow$ SL1 $\square$ ... $\square$ Bn $\rightarrow$ SLn od . We will use BB to abbreviate B1 or .. or Bn .

One reason of unsuccessful refinement of the scheme may be that the guards are not tolerant enough to establish R . If in this case it is indeed impossible to find more guarded commands, we have at least established P and not BB so that it only remains to invent a suitable refinement for "gvn P and not BB est R" . (However, as a particular case, P might imply not BB so that the repetition is equivalent to skip and no progress has been made at all!)

Other reasons might be remedied by inventing another T or even a new P .

Pragmatics Pr2 almost literally occurs in the last paragraph of chapter 6 "on the design of properly terminating constructs". It is also quite clearly used in chapter 8, in the third example, p. 57, the fourth example p. 61, and the first version of the fifth example, p. 62.

We now show the result of using Pr2 for the problems mentioned in examples E1 and E4 . For fairness of comparison, we choose the same P and T as Dijkstra does.

Ad example E1 . The relevant data are

R: f(k)$\geq$f(1..n-1) and 0$\leq$k<n

P: f(k)$\geq$f(1..j-1) and 0$\leq$k<j$\leq$n

T: n-j .

An obvious candidate to consider is j:=j+1 . We calculate and find

P and j$\neq$n and f(k)$\geq$f(j) $\Longrightarrow$ wp(--,P) and wdec(--,T) .

There are now two ways to proceed. The first one is to take the alternative

A11: j$\neq$n $\rightarrow$ "gvn j$\neq$n est f(k)$\geq$f(j)"; j:=j+1 ,

and fortunately we are through: do A11 od establishes R . It only remains to refine "gvn j$\neq$n est f(k)$\geq$f(j)" . This leads to the program

S1: do j$\neq$n $\rightarrow$ if f(k)$\geq$f(j) $\rightarrow$ skip $\square$ f(k)$\leq$f(j) $\rightarrow$ k:=j fi; j:=j+1 od .

The second way to proceed is as follows. Choose the alternative (note the change of $\underline{\text{and}}$ into $\underline{\text{cand}}$)

A12: $j \neq n\ \underline{\text{cand}}\ f(k) \geq f(j) \rightarrow j := j+1$.

Alas, $\underline{\text{do}}$ A12 $\underline{\text{od}}$ doesn't establish R . So we look for further alternatives, and with a little inspiration ("At least sometimes k must be assigned a value, and j is a good candidate") we find

A13: $j \neq n\ \underline{\text{cand}}\ f(k) \leq f(j) \rightarrow k, j := j, j+1$.

Fortunately we are through: R is established by

$$\begin{array}{ll} \text{S2: } \underline{\text{do}} & j \neq n\ \underline{\text{cand}}\ f(k) \geq f(j) \rightarrow j := j+1 \\ & \square\ j \neq n\ \underline{\text{cand}}\ f(k) \leq f(j) \rightarrow k, j := j, j+1 \\ & \underline{\text{od}}\ . \end{array}$$

Ad example E4 . The relevant data are

R: $allsix = \underline{A}\ i: 0..n-1.\ f(i) = 6$

P: $(allsix = \underline{A}\ i: 0..j-1.\ f(i) = 6)\ \underline{\text{and}}\ 0 \leq j \leq n$,

T: $n-j$.

An obvious candidate command is $j := j+1$. Calculation of wp and wdec justifies

A41: $j \neq n\ \underline{\text{cand}}\ f(j) = 6 \rightarrow j := j+1$.

Alas, $\underline{\text{do}}$ A41 $\underline{\text{od}}$ doesn't establish R , so we go on. At least some command must have the potential effect of allsix:=false . In that case any increase of j seems to keep P invariant.

Indeed, calculation show that

wdec(allsix,j:=false,n, T) = true,

wp(allsix,j:=false,n, P) =

= P[allsix,j ← false,n]

= $\underline{\text{not}}$ allsix $\underline{\text{or}}$ $\underline{E}$ i: j..n-1. $f(i) \neq 6$ provided P holds

$\Leftarrow j \neq n\ \underline{\text{and}}\ f(j) \neq 6$ provided P holds.

(The term $j \neq n$ is required on account of the range restriction in $\underline{E}$ i: j..n-1 and makes as well f(j) defined! And the condition is a weakest one in the sense that, assuming P holds, for any condition B satisfying $j \neq n\ \underline{\text{and}}\ f(j) \neq 6 \Longrightarrow B \Longrightarrow$ wp(allsix,j:=false,n, P), it holds true that $B \Longrightarrow j \neq n\ \underline{\text{and}}\ f(j) \neq 6$.)

So we choose

A42: $j \neq n\ \underline{\text{cand}}\ f(j) \neq 6 \rightarrow$ allsix,j:=false,n .

Fortunately we are through: R is established by

$$\begin{array}{ll} \text{S4: } \underline{\text{do}} & j \neq n\ \underline{\text{cand}}\ f(j) = 6 \rightarrow j := j+1 \\ & \square\ j \neq n\ \underline{\text{cand}}\ f(j) \neq 6 \rightarrow allsix, j := false, n \\ & \underline{\text{od}}\ . \end{array}$$

Taking an invariant relation P different from the one taken bij Dijkstra, we also describe the outcome of using pragmatics Pr2 for example E3 . Given is Y≥0, we take

R: $X^Y = Z$

P: $X^Y = Z * x^y$ and y≥0 ,

T: y .

The various candidates for a decrease of T are x,y:=z*x,y-1 , x,y:=x*x,y/2 , x,y:=x*x*x,y/3 and so on. Calculation of wp and wdec leads to

A31: y≠0 → z,y:=z*x,y-1 ,

A32: 2|y≠0 → x,y:=x*x,y/2 ,

A33: 3|y≠0 → x,y:=x*x*x,y/3 ,

and so on. The repetition

S3: do A31 ▯ A32 ▯ A33 ... od

establishes R !

In our opinion the programs derived by pragmatics Pr2 express the conceptual algorithm better than those derived by Pr1 , although, admittedly, they might need more adaption if they are intended for efficient execution on a sequential machine. Moreover, any program derived in any way should be subjected to a subsequent engineering phase after it has been formulated in a first form, and the better the conceptual algorithm has been expressed, the better -- we think -- it is suited for optimizing transformations.

Example, of typical transformation. Recall program S3: do A31 ▯ A32 ▯ A33 ... od . An obviously correct transformation yields

S3': do y≠0 {i.e. the guard of some A3i holds} →

... do A33 od; do A32 od; if A31 ▯ not y≠0 → skip fi

od .

Then, on account of the invariance of y≠0 over each A3i.command (for i>1), we may, in the second line of S3' , delete each test y≠0 (and replace not y≠0 by false).

Thus we have exploited the nondeterminacy by forcing a particular "flow through" S3 with the result that the most effective commands (say $x,y:=x^4,y/4$ and so on) are performed first and some test have become superfluous. (End of example.)

Suprisingly, repetitions derived by Pr2 may have another serious drawback. It might sometimes be the case that R has already been established although still some guards do hold and "repetition continues". Thus we define

Pr3: the same as Pr2 except that

```
    P and Bi ⟹ wp(SLi, P) and wdec(SLi, T) and not R .
```

Pragmatics Pr3 might be viewed as a combination of Pr1 and Pr2 . However the resulting program doesn't express the conceptual algorithm as cleas as possible, because the termination condition has been mixed up with the guards proper. What is needed is a construct which expresses both the various guarded commands and the termination condition separately; only such a construct gives the best information for subsequent optimizing transformations.

Remark. The notational ideal is approximated by use of recursive refinement (Hehner 76):

```
   "DO":
   if Q → skip □ B1 → SL1;"DO" □ ... □ Bn → SLn;"DO" fi .
```

implicitly

However, the above text expresses repetition very explicitly and hence unpleasantly if repetition is considered a language primitive.
(End of remark.)

Example. Compare the "allsix programs" E4 and S4 . If in the development of S4 we would not have invented the command allsix,j:=false,n but only allsix,j:=false,j+1 , we would have got

```
   do j≠n cand f(j)=6 → j:=j+1
    □ j≠n cand f(j)≠6 → allsix,j:=false,j+1
   od .
```

In this case P and Bi ⇏ not R . Indeed, the weakest Q such that P and Q ⟹ R is

Q: j=n or not allsix .

Thus in the above repetition each guard may be strengthened with not Q , yielding after simplication

```
S4': do allsix and j≠n cand f(j)=6 → j:=j+1
      □ allsix and j≠n cand f(j)≠6 → allsix,j:=false,j+1
     od .
```

The formulation with recursive refinement reads

```
S4": "DO": if j=n or not allsix → skip
           □ j≠n cand f(j)=6 → j:=j+1; "DO"
           □ j≠n cand f(j)≠6 → allsix,j:=false,j+1; "DO"
          fi .
```

(End of example.)

3. References

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

Hehner, E.C.R. (1976): do considered od, a contribution to the programming calculus, University of Toronto, revised 1978 January.