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
2. The pragmatics
3. References
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 successful 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 \( P \text{ and } Q \implies R \), and invent a variant function \( T \) (such that \( P \implies T \geq 0 \)).
Then try to refine the scheme
"est \( P \); do not \( Q \) \implies "gvn not \( Q \) mnt \( P \) dcr \( T \)" od.

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

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:
\[
\begin{align*}
do & j \neq n \rightarrow \text{a step towards } j = n \text{ under invariance of } P \od, \\
& \quad \text{if } f(k) \geq f(j) \rightarrow j := j + 1 \\
& \quad \quad \quad \text{if } f(k) \leq f(j) \rightarrow k, j := j, j + 1 \fi \od.
\end{align*}
\]

E2: second version of fifth example, p. 62; we quote:
\[
\begin{align*}
do & a + 1 \neq b \rightarrow \text{decrease } b - a \text{ under invariance of } P \od.
\end{align*}
\]

E3: sixth example, p. 65; we quote:
\[
\begin{align*}
do & h \neq 1 \rightarrow \text{squeeze } h \text{ under invariance of } P \od, \\
do & y \neq 0 \rightarrow 2/y \rightarrow x, y := x \times x, y/2 \od; \\
& \quad y, z := y - 1, z \times x \od.
\end{align*}
\]

E4: seventh example, p. 67; we quote:
\[
\begin{align*}
do & j \neq n \rightarrow \text{allsix :=allsix and } f(j) = 6; j := j + 1 \od.
\end{align*}
\]

E5: eight example, p. 70; we quote:
\[
\begin{align*}
do & s \neq r \rightarrow \{ \ldots \} \text{ increase } s \text{ by a suitable amount under } \\
& \quad \text{invariance of } P \{ \ldots \} \od.
\end{align*}
\]

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 \text{ and } Q \rightarrow R, \text{ and invent a variant function } T \text{ (such that } P \rightarrow T \geq 0 \).
\]

Then try to refine the scheme "est P"; do "mnt P dec T" od.
Thus we should invent some statement lists $SL_i$, and derive conditions $Bi$ such that for each $i$

$$P \text{ and } Bi \implies wp(SL_i, P) \text{ and } wdec(SL_i, T).$$

The repetition then reads $\text{do } B_1 \implies SL_1 \[ \ldots \] B_n \implies SL_n \text{ od }$. We will use $BB$ to abbreviate $B_1 \text{ or } \ldots \text{ or } B_n$.

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 $P \text{ and } \text{not } BB$ so that it only remains to invent a suitable refinement for "gvn $P \text{ and } \text{not } BB$ est $R"$. (However, as a particular case, $P$ might imply $\text{not } BB$ 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

$$P \text{ and } j \neq n \text{ and } f(k) \geq f(j) \implies wp(-,-,P) \text{ and } wdec(-,-,T).$$

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

$A1$: $j \neq n \implies \text{"gvn } j \neq n \text{ est } f(k) \geq f(j)"; j:=j+1$,

and fortunately we are through: $\text{do } A1 \text{ od }$ establishes $R$. It only remains to refine "gvn $j \neq n \text{ est } f(k) \geq f(j)"$. This leads to the program

$S1$: $\text{do } j \neq n \implies \text{if } f(k) \geq f(j) \implies \text{skip } [ f(k) \leq f(j) \implies k:=j \text{ fi}; j:=j+1 \text{ od }$.

The second way to proceed is as follows. Choose the alternative
(note the change of and into cand )

A12: \( j \notin \text{ cand } f(k) \geq f(j) \rightarrow j := j + 1 \).

Alas, do A12 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: \( j \notin \text{ cand } f(k) \leq f(j) \rightarrow k, j := j, j + 1 \).

Fortunately we are through: R is established by

S2: do \( j \notin \text{ cand } t(k) \geq t(j) \rightarrow j := j + 1 \)

\[ \begin{align*}
\text{if } j \notin \text{ cand } f(k) \leq f(j) & \rightarrow k, j := j + 1 \\
\text{od.} \end{align*} \]

Ad example E4. The relevant data are

R: \( \text{ alls } = A \_ i: 0..n-1. f(i) = 6 \)

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

T: \( n-j \).

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

A41: \( j \notin \text{ cand } f(j) = 6 \rightarrow j := j + 1 \).

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

Indeed, calculation show that

\( \text{ wdec(alls = false, n, T) = true, wp(alls = false, n, P) = } \)

\[ \begin{align*}
&= \text{ P[alls = false, P]}
&= \text{ not alls or } \text{ E i: j..n-1. f(i) \neq 6 provided } P \text{ holds}
\end{align*} \]

\[ \leq j \notin \text{ and } f(j) \neq 6 \text{ provided } P \text{ holds.} \]

(The term \( j \notin \) is required on account of the range restriction in \( E i: j..n-1 \) 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 \( j \notin \text{ and } f(j) \neq 6 \rightarrow B \rightarrow \text{ wp(alls = false, n, P), it holds true that } B \rightarrow j \notin \text{ and } f(j) \neq 6. \)

So we choose

A42: \( j \notin \text{ cand } f(j) \neq 6 \rightarrow \text{ alls = false, n.} \)

Fortunately we are through: R is established by

S4: do \( j \notin \text{ cand } f(j) = 6 \rightarrow j := j + 1 \)

\[ \begin{align*}
\text{ if } j \notin \text{ cand } f(j) \neq 6 & \rightarrow \text{ alls = false, n}
\text{ od.} \end{align*} \]
Taking an invariant relation \( P \) different from the one taken bij Dijkstra, we also describe the outcome of using pragmatics \( \text{Pr2} \) for example \( E3 \). Given is \( y \geq 0 \), we take
\[
R: x^y = z \\
P: x^y = z \times x^y \text{ and } y \geq 0 \\
T: y .
\]
The various candidates for a decrease of \( T \) are \( x, y = z \times x, y - 1 \), \( x, y = x \times x, y/2 \), \( x, y = x \times x \times x, y/3 \) and so on. Calculation of \( wp \) and \( \text{wdec} \) leads to
\[
A31: y \neq 0 \rightarrow z, y = z \times x, y - 1 \\
A32: 2 \mid y \neq 0 \rightarrow x, y = x \times x, y/2 \\
A33: 3 \mid y \neq 0 \rightarrow x, y = x \times x \times x, y/3 ,
\]
and so on. The repetition
\[
S3: \text{do } A31 \square A32 \square A33 \ldots \text{ od}
\]
establishes \( R \).

In our opinion the programs derived by pragmatics \( \text{Pr2} \) express the conceptual algorithm better than those derived by \( \text{Pr1} \), 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 \( S3: \)
\[
\text{do } A31 \square A32 \square A33 \ldots \text{ od} . \text{ An obviously correct transformation yields}\nS3': \text{do } y \neq 0 \{ \text{i.e. the guard of some } A3i \text{ holds} \} \rightarrow \\
\ldots \text{ do } A33 \text{ od}; \text{ do } A32 \text{ od}; \text{ if } A31 \square \text{ not } y \neq 0 \rightarrow \text{ skip fi} \\
\text{ud} .
\]
Then, on account of the invariance of \( y \neq 0 \) over each \( A3i\.command \)
(for \( i > 1 \)), we may, in the second line of \( S3' \), delete each test \( y \neq 0 \) (and replace \( \text{not } y \neq 0 \) by \( \text{false} \)).

Thus we have exploited the nondeterminacy by forcing a particular "flow through" \( S3 \) with the result that the most effective commands (say \( x, y = x \times 4, y/4 \) 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 \implies 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 clearly 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":
   \textbf{if} \ Q \ + \ \textbf{skip} \ [ \ B1 \ + \ SL1;"DO" \ [ \ \ldots \ [ \ Bn \ + \ SLn;"DO" \ fi.}
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\neq n \ \textbf{cand} \ f(j)=6 \ \rightarrow \ j:=j+1
\ [ \ j\neq n \ \textbf{cand} \ f(j)\neq 6 \ \rightarrow \ allsix,j:=false,j+1
\ od.
In this case P and Bi \implies not R. Indeed, the weakest Q such that
P and Q \implies R is
Q: j=n or not allsix.
Thus in the above repetition each guard may be strengthened with not Q,
yielding after simplification
S4': do allsix and j\neq n \ \textbf{cand} \ f(j)=6 \ \rightarrow \ j:=j+1
\ [ \ allsix and j\neq n \ \textbf{cand} \ f(j)\neq 6 \ , \ allsix,j:=false,j+1
\ od.
The formulation with recursive refinement reads

\[ S4^* : \text{"DO" if } j = n \text{ or allsix } \rightarrow \text{skip} \]
\[ \begin{align*}
\text{if } j \neq n \text{ cand } f(j) &= 6 \rightarrow j := j + 1; \text{ "DO" } \\
\text{if } j \neq n \text{ cand } f(j) \neq 6 \rightarrow \text{allsix, } j := \text{false, } j + 1; \text{ "DO" } \\
\end{align*} \]

(End of example.)

3. References
