Privé

TECHNISCHE HOGESCHOOL TWENTE

ONDERAFDELING DER TOEGEPASTE WISKUNDE

DISCIPLINED CONSTRUCTIONS OF PARTITION

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

September 1978

Abstract. Various formal developments, fully in the spirit of Dijkstra's "A discipline of programming", and subsequent optimizing program transformations for the algorithm PARTITION (Hoare 61) are described and analyzed.

Keywords and phrases. Programming methodology, guarded commands, invariant relation, variant function, repetitive construct, nondeterminacy, program transformation.

Contents.

1. Introduction and conclusions

In this paper we describe and analyze various ways the algorithm PARTITION (Hoare 61) could have been constructed.

The characteristics of this paper are the following.

(1) All developments take place by quite formal manipulations, minimizing the need for inspiration, invention, intuition or what you may call it, and fully in the spirit of "A discipline of programming"(Dijkstra 76).

(2) Unlike (Dijkstra 76) we also describe some (tricky ?) program transformations. But needless to say, they are formally treated as well. Also, we extract the general principles behind them. (Sections 5-7).

(3) We explicitly distinguish two pragmatics for the repetive construct. (Sections 3 and 4).

The following conclusions may be drawn.

(4) Some quite satisfactory programs are constructed with very little inspiration needed. (Sections 3 and 4)

(5) Dijkstra's repetitive construct allows several solutions which are not easily described with the usual while-statement. (This conclusion is meant for the uninitiated reader only). (Sections 3 and 4)

(6) The program structure mostly occurring in the literature is obtained from the simpler guarded command version by "restricting nondeterminacy". (Section 6)

(7) The invariant relation prompted by the program specification differs from the one mostly used in the literature but leads to equally satisfactory programs. (Section 3)

(8) Van Emden's version arises more naturally than Hoare's original version, (Van Emden 71)(Hoare 61). (Section 8)

2. The program specification

Given an integer array z and integer constants m,n satisfying $z.lob \le m \le n \le z.hib$, and integer variables l,r, it is the purpose of the program to permute the array z(m..n) and partition it into a leftpart $z(m..l)$ of small elements, a rightpart z(r..n) of large elements, and a middle part $z(l+1..r-1)$ of equal elements in value in between the small and large ones. Either of the parts may be empty, but neither the left part nor the right part may be the full array z(m..n) (hence m..n must be nonempty).

In the development of the algorithm it has appeared that we only need z:swap as assignment operation to z . Hence the final value is a permutation of the initial value. For simplicity we will use this knowledge in advance and formulate the following constraint.

C: z:swap is the only value changing operation allowed on z .

(The remainder of) the relation to be established is easily formalized as follows.

R: $m-1 \le l < n \ \underline{\text{and}}\ m < r \le n+1 \ \underline{\text{and}}\ l < r \ \underline{\text{and}}$

$\quad \underline{\text{E}}\nu.\ z(m..l) \le \nu \le z(r..n) \ \underline{\text{and}}\ \nu = z(l+1..r-1)$.

It is not requested whether the segment $l+1..r-1$ should be minimal, i.e. empty, or maximal, i.e. both $l:=l-1$ and $r:=r+1$ will certainly disturb R . Any such request can be dealt with after the establishment of R .

3. Developments maintaining $l<r$

Relation R strongly suggests a candidate for an invariant relation: drop the term $\nu = z(l+1..r-1)$. What remains is easily established by $l,r:=m-1,n+1$. The difference between r and l seems a good candidate for the variant function. On account of the invariant relation the difference is bounded below by one; so we substract one from the difference in order to obtain lower-bound zero. Thus we obtain the invariant relation P1 and variant function T1 :

P1: $m-1 \le l < n \ \underline{\text{and}}\ m < r \le n+1 \ \underline{\text{and}}\ l < r \ \underline{\text{and}}\ z(m..l) \le z(r..n)$,

T2: $r-l-1$.

We will now try to develop the repetition <u>do</u> "mnt P1 dcr T1" <u>od</u> (this abbreviates <u>do</u> "maintain P1 and decrease T1" <u>od</u>).

Let us consider $l:=l+1$; it is an obvious candidate for a decrease of T1 , the invention of which doesn't require too much inspiration. We compute wp($l:=l+1$, P1) and wdec($l:=l+1$, T1) .

wp($l:=l+1$, P1)=

$\quad = P1[l \leftarrow l+1]$

$\quad = m-1 \le l+1 < n \ \underline{\text{and}}\ m < r \le n+1 \ \underline{\text{and}}\ l+1 < r \ \underline{\text{and}}\ z(m..l+1) \le z(r..n)$

$\quad = l+1 \ne n \ \underline{\text{and}}\ l+1 \ne r \ \underline{\text{and}}\ z(l+1) \le z(r..n)$, provided P1 holds

$\quad = n \ne l+1 \ne r \ \underline{\text{and}}\ z(l+1) \le z(r..n)$, provided P1 holds.

For wdec we follow (Dijkstra 76) p. 43.

wdec($l:=l+1$, T1) =

$= tmin \le T1-1$ where $tmin = \underline{min}\ t0.\ wp(l{:}=l+1,\ T1 \le t0)$

$=$ " " $tmin = \underline{min}\ t0.\ r-l-2 \le t0$

$=$ " " $tmin = r-l-2$

$= r-l-2 \le r-l-2$

$=$ true .

Thus we find as a suitable "step, decreasing T1 while maintaining P1 ", A1 and analoguously A2 :

A1: $n \ne l+1 \ne r$ <u>cand</u> $z(l+1) \le z(r..n) \rightarrow l{:}=l+1$,

A2: $l \ne r-1 \ne m$ <u>cand</u> $z(m..l) \le z(r-1) \rightarrow r{:}=r-1$.

Note that we have replaced <u>and</u> by <u>cand</u> in order to make the evaluation of the second term well defined.

Not surprisingly, the alternatives A1 and A2 are not sufficient; <u>do</u> A1 ▯ A2 <u>od</u> does not establish R . Therefore we look for further alternatives decreasing T1 while maintaining P1 . It is left to the reader to imagine that z:swap(l+1,r-1); l;r:=l+1,r-1 <u>may</u> do when neither of the guards of A1 and A2 holds. Formal calculation shows:

$wp(z{:}swap(l+1,r-1);\ l,r{:}=l+1,r-1\ ,\ P1) =$

$= wp(z{:}swap(l+1,r-1),\ wp(l,r{:}=l+1,r-1\ ,\ P1))$

$= wp(z{:}swap(l+1,r-1),\ P1[l,r \leftarrow l+1,r-1])$

$= P1[l,r \leftarrow l+1,r-1][z \leftarrow z']$

here and in what follows z' is defined by

$z'(i) = \underline{if}\ i=l+1 \rightarrow z(r-1)\ \square\ i=r-1 \rightarrow z(l+1)\ \square\ l+1 \ne i \ne r-1 \rightarrow z(i)\ \underline{fi}$

$= m-1 \le l+1 < n$ <u>and</u> $m < r-1 \le n+1$ <u>and</u> $l+1 \le r-1$ <u>and</u> $z'(m..l+1) \le z'(r-1..n)$

$= r \ne l+1 \ne r-1 \ne l$ (provided P1 holds) <u>and</u> $z'(m..l+1) \le z'(r-1..n)$

$= 2 \ne r-l \ne 1$ <u>and</u> $(z'(m..l),\ z'(l+1)) \le ((z'(r-1),\ z'(r..n))$

$= 2 \ne r-l \ne 1$ <u>and</u> $z(m..l) \le z(l+1) \ge z(r-1) \le z(r..n)$ provided P1 holds.

Secondly,

$wdec(z{:}swap(l+1,r-1);\ l,r{:}=l+1,r-1\ ,\ T1) =$

$= tmin \le T1-1$ where $tmin = \underline{min}\ t0.\ wp(\sim\sim\sim,\ T1 \le t0)$

$=$ " " $tmin = \underline{min}\ t0.\ (T1 \le t0)[l,r \leftarrow l+1,\ r-1][z \leftarrow z']$

$=$ " " $tmin = \underline{min}\ t0.\ (r-1)-(l+1)-1 \le t0$

$=$ " " $tmin = r-l-3$

$= r-l-3 \le r-l-2$

$=$ true .

Hence a suitable alternative is

A3: $2 \ne r-l \ne 1$ <u>cand</u> $z(m..l) \le z(l+1) \ge z(r-1) \le z(r..n) \rightarrow$

$z{:}swap(l+1,r-1);\ l,r{:}=l+1,r-1$.

Unfortunately we are still not through. With the initialization $l,r:=m-1,n+1$ the repetition $\underline{\text{do}}$ A1 □ A2 □ A3 $\underline{\text{od}}$ doesn't establish R . Indeed, all guards may be false and R needn't hold if $l+1=n$. (This situation can occur as follows. After the initialization $r..n$ is empty, so $z(l+1)\le z(r..n)$ holds for all l , and the first alternative may be executed until $l+1=n$.) It seems quite hard to add another alternative to handle this case. (Try it) However, if $m<n$ the nasty emptyness of $m..l$ and $r..m$ may easily be avoided by the following initialization:

$\underline{\text{if}}\ z(m)\le z(n) \rightarrow \text{skip}\ \square\ z(m)\ge z(n) \rightarrow z{:}swap(m,n)\ \underline{\text{fi}};$

$l,r:=m,n$ {note that $l<r$ requires $m<n$}.

Indeed, $m\le l$ $\underline{\text{and}}$ $r\le n$ is kept invariant independently of the guards, and on account of $l<r$ we may simplify the guards so as to obtain the repetition S1 and invariant relation P2 :

P2: P1 $\underline{\text{and}}$ $m\le l$ $\underline{\text{and}}$ $r\le n$, or simplified,

$m\le l<r\le n$ $\underline{\text{and}}$ $z(m..l)\le z(r..n)$,

S1: $\underline{\text{do}}$ A4 □ A5 □ A6 $\underline{\text{od}}$,

A4: $l+1\ne r$ $\underline{\text{cand}}$ $z(l+1)\le z(r..u) \rightarrow l:=l+1$;

A5: $l\ne r-1$ $\underline{\text{cand}}$ $z(m..l)\le z(r-1) \rightarrow r:=r-1$,

A6: $2\ne r-l\ne 1$ $\underline{\text{cand}}$ $z(m..l)\le z(l+1)\ge z(r-1)\le z(r..n) \rightarrow$

$z{:}swap(l+1,r-1);\ l,r:=l+1,r-1$.

The above initialization and repetition establish R .

Proof. On account of $m\le l$ $\underline{\text{and}}$ $r\le n$

(1) the segments $m..l$ and $r..n$ are nonempty.

Now assume $l+1\le r-1$. On account of the falsity of all guards, we find from A4 and A5 respectively:

(2a,b) $z(l+1)\not\le z(r..n)$, or on account of P2 and (1), $z(m..l)<z(l+1)$,

(3a,b) $z(m..l)\not\le z(r-1)$, or on account of P2 and (1), $z(r-1)<z(r..n)$,

From A6 there arise four possibilities:

either: $l+1=r-1$ but with (2b) this contradicts (3a),

or: $z(m..l)\not\le z(l+1)$ but this contradicts (2b),

or: $z(l+1)<z(r-1)$ but with (2b) this contradicts (3a),

or: $z(r-1)\not\le z(r..n)$ but this contradicts (2b).

Hence all possibilities lead to a contradiction. Therefore $l+1\not\le r-1$ or, equivalently, $l+1..r-1$ is empty; in this case P2 implies R .

(End of proof.)

Remark. In view of the invariant relation P2 , the conditions $l+1 \neq r$, $l \neq r-1$, $2 \neq r-l \neq 1$ are equivalent with respectively $l+1<r$, $l<r-1$, $l+1<r-1$. However, in general the latter are stronger than the former, and robustness decreases if we replace any of the former by the corresponding one of the latter. Indeed, if accidentally the program is executed with not $m<n$, then fortunately S1 will not terminate properly, whereas it will terminate (consequently with unreliable results) if any of the stronger conditions has been used. (End of remark.)

* * *

Above we had the strategy to develop the repetition do "mnt P2 dcr T1" od . We will now take another strategy. Note that P2 and $r-l=1$ implies R . Thus we try to develop the following repetition:

do $r-l \neq 1 \rightarrow$ "gvn $r-l \neq 1$ mnt P2 dcr T1" od .

Formal calculations show:

$wp(l:=l+1,\ P2) = z(l+1) \leq z(r..n)$ provided P2 and $r-l \neq 1$ holds,

$wp(r:=r-1,\ P2) = z(m..l) \leq z(r-1)$ provided P2 and $r-l \neq 1$ holds,

$wp(z{:}swap(l+1,r-1);\ l,r:=l+1,\ r-1\ ,\ P2) =$

$l+1 \neq r-1$ and $z(m..l) \leq z(l+1) \geq z(r-1) \leq z(r..n)$ provided P2 and $r-l \neq 1$ holds,

The wdec of any of these statements with respect to T1 equals true , and moreover if $r-l \neq 1$ then certainly one of the conditions will hold. Thus the repetition may read

```
S2: do r-l≠1 →
        if z(l+1)≤z(r..n) → l:=l+1
        ▯ z(m..l)≤z(r-1) → r:=r-1
        ▯ l+1≠r-1 and z(m..l)≤z(l+1)≥z(r-1)≤z(r..n) →
              z:swap(l+1,r-1); l,r:=l+1,r-1
        fi
    od .
```

Note that the proof of the establishment of R is now replaced by essentially the same proof of proper termination of the alternative construct.

Remark 1. We can make the program more robust by replacing the term $l+1 \neq r-1$ in the third guard by the (stronger) term $l+1<r-1$. Indeed, the disallowed initial state satisfying $m=n$ might lead to proper termination with unreliable results (if e.g. $z \cdot low<m=n<z \cdot hib$) in S2 , whereas it leads to abortion of program execution with the proposed change. (End of remark.)

Remark 2. In a subsequent optimization phase the term $l+1 \neq r-1$ may even be deleted, provided the main guard is replaced by $r-l>1$ (which unfortunately detracts from robustness) and the repetition is continued with

$\underline{\text{if}}$ $l=r \rightarrow$ either $l:=l-1$ or $r:=r+1$ (or even both)

$\square$ $l<r \rightarrow$ skip

$\underline{\text{fi}}$.

The easily verifiable invariant relation then reads

$P2$ $\underline{\text{or}}$ $(l=r$ $\underline{\text{and}}$ $z(m..l) \leq z(r..n))$.

In section 7 this transformation is treated in a general setting. (End of remark.)

4. Developments establishing $l<r$

With some intelligence and inspiration, we may proceed to transform relation R into an equivalent but differently written relation, as follows.

$l<r$ $\underline{\text{and}}$ $E\nu.$ $z(m..l) \leq \nu \leq z(r..n)$ $\underline{\text{and}}$ $\nu = z(l+1..r-1)$

$=$ $l<r$ $\underline{\text{and}}$ $z(m..l) \leq z(r..n)$ $\underline{\text{and}}$ $z(m..l) \leq z(l+1..r-1) = z(l+1..r-1) \leq z(r..n)$

$=$ $l<r$ $\underline{\text{and}}$ $z(m..r-1) \leq z(l+1..n)$.

Thus the full relation to be established now becomes

R: $m-1 \leq l<n$ $\underline{\text{and}}$ $m<r \leq n+1$ $\underline{\text{and}}$ $l<r$ $\underline{\text{and}}$ $z(m..r-1) \leq z(l+1..n)$.

The above relation, although equivalent to the original R , suggests a quite different invariant: drop the term $l<r$, yielding

P3: $m-1 \leq l<n$ $\underline{\text{and}}$ $m<r \leq n+1$ $\underline{\text{and}}$ $z(m..r-1) \leq z(l+1..n)$.

This relation is easily established by

$\underline{\text{if}}$ $z(m) \leq z(n) \rightarrow$ skip $\square$ $z(m) \geq z(n) \rightarrow$ z:swap(m,n) $\underline{\text{fi}}$;

$r,l:=m+1,n-1$.

Note that it is not required that $m \neq n$. (Note also that now l must grow to the left and r to the right.) The difference between l and r seems a good candidate for the variant function. On account of the invariant relation the difference is bounded below in case $l=m-1$ and $r=n+1$;

addition of a constant, so that the lowerbound becomes zero, yields
T2: $n-m+\ell-r$.

Similarly to the previous section, we first try to develop
<u>do</u> "mnt P3 dcr T2" <u>od</u> .
Obvious candidates for a decrease of T2 are the statements $\ell:=\ell-1$ and $r:=r+1$ and $z{:}swap(r,\ell);\ r,\ell:=r+1,\ell-1$. Formal calculation shows

$wp(\ell:=\ell-1,\ P3) = m-1\neq\ell$ <u>and</u> $z(m..r-1)\leq z(\ell)$ provided P3 holds,
$wp(r:=r+1,\ P3) = r\neq n+1$ <u>and</u> $z(r)\leq z(\ell+1..n)$ provided P3 holds,
$wp(z{:}swap(r,\ell);\ r,\ell:=r+1,\ell-1\ ,\ P3) =$
    $r\leq\ell$ <u>and</u> $z(m..r-1)\leq z(r)\geq z(\ell)\leq z(\ell+1..n)$ <u>or</u>
    $m-1\neq\ell<r\neq n+1$ <u>and</u> $z(r)=z(\ell)$ provided P3 holds.

For each of these statements S , wdec(S, T2)=true . Thus we arrive at the following alternatives.

A7: $m-1\neq\ell$ <u>cand</u> $z(m..r-1)\leq z(\ell)\ \rightarrow\ \ell:=\ell-1$,
A8: $r\neq n+1$ <u>cand</u> $z(r)\leq z(\ell+1..n)\ \rightarrow\ r:=r+1$,
A9 $r\leq\ell$ <u>cand</u> $z(m..r-1)\leq z(r)\geq z(\ell)\leq z(\ell+1..n)\ \rightarrow$
    $z{:}swap(r,\ell);\ r,\ell:=r+1,\ell-1$.

(Rather arbitrarily we have omitted the disjunct $m-1\neq\ell<r\neq n+1$ <u>cand</u> $z(\ell)=z(r)$ in the guard of A9 . The omission gives a simpler text; the omitted effect may be obtained by A7 and A8 instead.) Fortunately we are through.
After the initialization, the repetition
S3: <u>do</u> A7 □ A8 □ A9 <u>od</u>
establishes $\ell<r$ (check!) hence R .

* * *

Above we have developed the scheme <u>do</u> "mnt P3 dcr T2" <u>od</u> . We will now describe the development of another scheme. First note that P3 <u>and</u> $\ell<r$ implies R . Thus choose the scheme
<u>do</u> $\ell\not<r\ \rightarrow$ "gvn $r\leq\ell$ mnt P3 dcr T2" <u>od</u> .

Formal calculation shows that

$wp(\ell:=\ell-1,\ P3) = z(m..r-1)\leq z(\ell)$ provided P3 <u>and</u> $r\leq\ell$ holds,
$wp(r:=r+1,\ P3) = z(r)\leq z(\ell+1..n)$ provided P3 <u>and</u> $r\leq\ell$ holds,
$wp(z{:}swap(r,\ell);\ r,\ell:=r+1,\ell-1,\ P3) =$
  $z(m..r-1)\leq z(r)\geq z(\ell)\leq z(\ell+1..n)$ provided P3 <u>and</u> $r\leq\ell$ holds.

Further, P3 <u>and</u> $r\leq\ell$ implies that at least one of these conditions holds.

Thus we obtain the repetition

```
S4: do r≤l →
        if z(m..r-1)≤z(l) → l:=l-1
        □  z(r)≤z(l+1..n) → r:=r+1
        □  z(m..r-1)≤z(r)≥z(l)≤z(l+1..n) → z:swap(r,l); r,l:=r+1,l-1
        fi
    od  .
```

Note that again the proof of proper termination of the alternative construct takes the place of essentially the same proof of the establishment of R by repetition S3 .

* * *

The two pragmatics of the repetitive construct have led to remarkably different programs: S3 establishes a maximal difference between l and r , whereas S4 establishes a minimal difference. Using recursive refinement (Hehner 76), one naturally obtains a program which nondeterministically establishes any difference between those two extremes:

```
S5: "mnt P3 est R":
    if l<r → skip
    □  m-1≠l cand z(m..r-1)≤z(l) → l:=l-1, "mnt P3 est R"
    □  r≠n+1 cand z(r)≤z(l+1..n) → r:=r+1; "mnt P3 est R"
    □  r≤l   cand z(m..r-1)≤z(r)≥z(l)≤z(l+1..n) →
               z:swap(r,l); r,l:=r+1,l-1; "mnt P3 est R"
    fi  .
```

Note that the correctness arguments of each of the alternatives has been given in the development of either S3 or S4 or both. (Termination is guaranteed because T2 is decreased before any of the semi-recursive calls.) Robustness may be increased by replacing $m-1 \neq l$ by $m \leq l$ and $r \neq n+1$ by $r \leq n$.

* * *

In the literature, the development of PARTITION mostly starts with the specification as given in section 2 and then proceeds with the invariant relation of this section, motivating the change in formulation of R by "inspiration". In the preparation of this paper, I did so as well. However, I was prompted to apply the formal machinery to the original formulation of R , avoiding the need for some "inspiration", and thus discovered the invariant relation and repetitions of section 3. After all, I find them as satisfactory as those of this section. Thus once again there is evidence that formal program construction may yield quite satisfactory results.

5. "Wirth's trick" as an optimizing program transformation

We now describe an optimizing program transformation found in (Wirth 76). It is applicable both to developments maintaining l<r and to those establishing l<r .

Recall from section 3 repetition S1 maintaining P2 .

P2: m≤l<r≤n <u>and</u> z(m..l)≤z(r..n) ,

S1: <u>do</u> A4 □ A5 □ A6 <u>od</u> ,

A4: l+1≠r <u>cand</u> z(l+1)≤z(r..n) → l:=l+1 ,

A5: l≠r-1 <u>cand</u> z(m..l)≤z(r-1) → r:=r-1 ,

A6: 1≠r-l≠2 <u>cand</u> z(m..l)≤z(l+1)≥z(r-1)≤z(r..u) →
      z:swap(l+1,r-1); l,r:=l+1,r-1 .

Now strengthen the guard of A4 into z(l+1)<z(r..n) . Then l+1≤r is an additional invariant relation of that guarded command (because r..n is nonempty), and it is already invariant over A5 (independently of its guard) and A6 , and it is also initially true with l,r:=m,n (because m<n is already assumed). Analoguously the guard of A5 may be strengthened. Thus we obtain the alternatives

A4': z(l+1)<z(r..n) → l:=l+1 ,

A5': z(m..l)<z(r-1) → r:=r-1 .

Although two guards have been strengthened, repetition

S1'' : <u>do</u> A4' □ A5' □ A6 <u>od</u>

still establishes the emptyness of l+1..r-1 hence R .

Similarly we may change S3 which maintains P3 .

P3: m-1≤l<n <u>and</u> m<r≤n+1 <u>and</u> z(m..r-1)≤z(l+1..n) ,

S3: <u>do</u> A7 □ A8 □ A9 <u>od</u>

A7: m-1≠l <u>cand</u> z(m..r-1)≤z(l) → l:=l-1 ,

A8: r≠n+1 <u>cand</u> z(r)≤z(l+1..n) → r:=r+1 ,

A9: r≤l cand z(m..r-1)≤z(r)≥z(l)≤z(l+1..n) →
        z:swap(r,l); r,l:=r+1,l-1 .

Now strengthen the guard of A7 into z(m..r-1)<z(l) . Then m≤l is an additional invariant relation of that guarded command (because m..r-1 is nonempty) and it is already invariant over A8 (independently of its guard) and A9 , and it is also initially true if (and only if) we assume m<n . Analoguously the guard of A8 may be strengthened. Thus we obtain

A7': z(m..r-1)<z(l) → l:=l+1 ,

A8': z(r)<z(l+1..n) → r:=r-1 .

Although the guards have been strengthened, repetition

S3': do A7' ▯ A8' ▯ A9 od

still establishes l<r hence R .

6. Optimization by restricting nondeterminacy

For simplicity we only consider repetition S3. Recall

S3: do A7 ▯ A8 ▯ A9 od ,

A7: m-1≠l cand z(m..r-1)≤z(l) → l:=l+1 ,

A8: r≠n+1 cand z(r)≤z(l+1..n) → r:=r+1 ,

A9: r≤l cand z(m..r-1)<z(r)≥z(l)≤z(l+1..n) → z:swap(r,l); r,l:=r+1,l-1 .

By bringing in more determinacy into the nondeterministic repetition over A7, A8 and A9 , we improve efficiency in that the evaluation of some terms of the guards is made superflouos.

First, note that not (A7.guard or A8.guard) and r≤l implies A9.guard . Therefore we "group together the potential steps over A7 and A8 ", so that thereafter in (a single, potential execution of) A9 the term z(m..r-1)≤.≥.≤... is superflouos. This transformation yields:

```
do A7.guard or A8.guard or A9.guard →
    do A7 ▯ A8 od;
    if r≤l → z:swap(r,l); r,l:=r+1,l-1 ▯ l<r → skip fi
do .
```

Second, note that the main guard of the above repetition is at least as weak as r≤l , whereas the latter is already sufficient for the establishment of R , Thus strengthen the main guard into r≤l .

Third, the relation not A7.guard is invariant over A8 , and not A8.guard is invariant over A7 . Therefore the inner repetition may be particularized into do A7 od; do A8 od .

All together this yields

```
S3": do r≤l →
        do A7 od; do A8 od;
        if r≤l → z:swap(r,l); r,l:=r+1,l-1 ▯ l<r → skip fi
     od  .
```

The above transformation may be combined with "Wirth's trick to yield

```
S3''': do r≤l →
          do A7' od; do A8' od;
          if r≤l → z:swap(r,l); r,l:=r+1,l-1 ▯ l<r → skip fi
       od  .
```

Also, the transformation is applicable to S1 (and S1') yielding S1" (and S1'''):

```
S1": do r-l≠1 →
        do A4 od; do A5 od;
        if 2≠r-l≠1 → z:swap(l+1,r-1); l,r:=l+1,r-1
        ▯ r-l=1 → skip
        fi
     od  .
```

Remark 1. The transformation seems not applicable to, or at least not so easy to describe for, S2 and S4 . A conclusion therefore might be that the pragmatics "develop do "mnt P dcr T" od" leads to repetitions better suitable for subsequent transformations. (End of remark.)

Remark 2. Repetition S3" (or S3''') often appears in the literature, always written with while do and if then. In my opinion, S3 is more fundamental than S3" , and certainly for educational and didactical purposes S3 is to be preferred over S3" . Indeed, any programmer developping S3" must at least have made (but possibly unconsciously) the reasoning as described in the development of S3 , and in particular must have thought (but possibly unconsciously) of A7, A8 and A9 as the basic alternative steps "towards termination while maintaining the invariant". (End of remark.)

Remark 3. The development of S3" from S3 clearly explains why the invariant holds again at some intermediate points inside the repeatable statement: in S3" the alternatives A7, A8 and A9 occur in sequential composition but each of them has been designed so as to maintain the invariant. (End of remark.)

Remark 4. In a subsequent optimization phase, we may even replace in S3" (and S3''') the construct

```
if r≤l → z:swap(r,l); r,l:=r+1,l-1 □ l<r → skip fi
```

by z:swap(r,l); r,l:=r+1,l-1

provided afterwards we undo a possible unwanted swap. Thus we obtain

```
S3*: do r≤l →
          do A7 od; do A8 od;
          z:swap(r,l); r,l:=r+1,l-1
     od;
     if r-1≤l+1 → skip □ l+1<r-1 → l,r:=l+1,r-1; z:swap(l,r) fi .
```

The invariant relation then reads

Q: P3 or

l+1<r-1 and (z(m..l-1),z(r),z(l+1..r-1))≤(z(l+1..r-1),z(l),z(r+1..n)).

It is easy to verify its initial establishment and its invariance, and to see that the final alternative construct establishes P3 and l<r from the repetitions postcondition Q and l<r .

Similarly for S1" and S1''' provided the main guard is replaced by r-l>1 . In section 7 we treat this transformation in a more general setting. (End of remark.)

7. A general treatment of some (tricky?) transformations

In this section we give a general and abstract treatment of the program transformations mentioned in remark 4 of section 6. Each of them transforms the repeatable statement of a repetitive construct so that only the very last step of the repetition is possibly affected and so that the unwanted effect can be easily undone afterwards.

The inspiration and motivation for such transformations has been got as follows. In an introductory programming course we teach the students to avoid conditional statements inside repeatable statements if their condition can only be valid at the (very first or) very last step of the repetition. For instance, one shouldn't write

```
i:=o;
while i≠n
   do if i=o then sum:=o;
      i:=i+1; sum:=sum+t(i);
      if i=n then print (sum)
   od .
```

Similarly, I find it not elegant and not efficient that in S3', S3", S1' and S1" the guard of the swap-command can only be invalid at the very last step of the repetition.

The formal justification below of the transformations doesn't prove the correctness of the modified programs from scratch, but uses the original correctness proof essentially. It also closely follows the intuitive argument that "the modification only possibly affects the very last step of the repetition and the unwanted effect is undone by the final alternative construct".

In the sequel {A}B{C} abbreviates A ⟹ wp(B,C) .

Theorem. Consider a repetitive construct of the form
S: do {P} C → SL0; {X} if B1 → SL1 □ B2 → SL2 fi {P} od
with the following properties.
First, P is an invariant relation and X holds at the point indicated:
(1) {P and C} SL0 {X and (B1 or B2)},
(2) {X and B1} SL1 {P},
(3) {X and B2} SL2 {P}.
Second, if the alternative construct is executed when B2 holds, then the repetition terminates, even if the whole alternative construct is replaced by SL1 :
(4) {X and B2} SL1 {not C},
(5) {X and B2} SL2 {not C}.
Third, if the alternative construct has been replaced by SL1 , it is decidable afterwards whether SL1 has been executed rightly or wrongly, by testing mutually exclusive conditions B1' and B2' :
(6) {X and B1} SL1 {C or B1'},
(7) {X and B2} SL1 {B2'},
(8) B1' and B2' = false.
And let $SL1^{-1}$ be any statement which undoes the effect of the "wrongly" execution of SL1 in a state X and B2 :
(9) {X and B2} SL1; $SL1^{-1}$ {X and B2} .
And assume finally
(10) the initial establishment of P also establishes C or B1' .
Then the postassertion P and not C of S is as well established by the optimized program

S': do C → SL0; SL1 od; if B1' → skip ▯ B2' → $SL1^{-1}$; SL2 fi .

Proof. Let Q be defined by

Q: (P and (C or B1')) or (QQ and C and B2')

where QQ is any relation which satisfies

(11) {X and B2} SL1 {QQ} SL^{-1} {X and B2} .

(By virtue of (9) such QQ exist.) Then Q is an invariant of S' .

(proof. At entrance of the loop Q and C implies P . Then by (1), X and (B1 or B2) holds just before SL1 . By (2) and (6) we find {X and B1} SL1 {P and (C or B1')} , and by (9), (5) and (7) we find {X and B2} SL1 {QQ and C and B2'} . Hence Q is reestablished upon exit from the repeatable statement.)

From the postassertion Q and not C the relation P and not C is established by the final alternative construct.

(proof. Using (8) it is obvious for the first alternative, and for the second one we use (8) then (11), (3) and (4).)

The initial establishment of Q follows from (10). (End of proof.)

As an example we apply the theorem to program S3" and obtain S3* as already shown in remark 6.4 . Recall

```
S3": do r ≤ l →
        do A7 od; do A8 od;
        if r ≤ l → z:swap(r,l); r,l:=r+1,l-1 ▯ l<r → skip fi
     od .
```

So it is obvious how to define C, SL0, B1, B2, SL1 and SL2 in order that

S3" = do C → SL0; if B1 → SL1 ▯ B2 → SL2 fi od .

Note that X is the invariant relation P3 . We choose

B1' : r+1≤l-1 ,

B2' : l-1<r+1 ,

$SL1^{-1}$: l,r:=l+1,r-1; z:swap(l,r) .

It is now very easy to verify conditions (1)-(10) of the theorem; therefore the program

S3*: do C → SL0; SL1 od; if B1' → skip ▯ B2' → $SL1^{-1}$; SL2 fi

is correct as well.

Remark. Note that we even need not know the invariant relation of S3* . The parts of Q necessarily known due to the verification of conditions (1)-(10) are P, C, B1' and B2' , but not QQ . Indeed, condition (9) can be proved without implicitly deriving QQ , as follows. First, any predicate P , on which SL1 will properly terminate, is invariant over SL1; $SL1^{-1}$ (which may be proved by textual manipulation). Second, from (3) it follows that SL1 will properly terminate on X and B2 . (End of remark.)

The intuitively similar transformation of remark 2 in section 3 can't be proved by the above theorem. True, program S2 can be brought into the required form by trivial textual manipulations, but no mutually exclusive relations B1' and B2' can be found, and quite essentially, $z(m..l) \leq z(l+1) \geq z(r-1) \leq z(r..n)$ implies, if $l+1 = r-1$, both the first and second guard. The statement of a theorem for this case is "left to the reader".

It is left open for discussion whether the transformations are "tricky" or not. It might be argued that they are not, because they are justified by such general theorems. However, it might as well be argued that they are, because the statement of the theorem is so lengthy and because the need for two different theorems for two similar cases suggests that no general principle is involved.

8. Further implementation.

Only the array comparisons, like $z(m..r-1) \leq z(l)$ and so on, need to be implemented further.

The most obvious way is to introduce two variables leftmax and rightmax which invariantly satisfy, e.g.,

leftmax = max z(m..r-1)

(or leftmax = max(z(m..r-1),-inf) if m..r-1 can be empty).

The condition $z(m..r-1) \leq z(l)$ may then be represented by leftmax $\leq z(l)$ and so on. The additional invariant relations are easily established and kept invariant as well.

Another possibility, in our view requiring more inspiration, is to choose a constant value ν , invariantly satisfying, e.g.,

$z(m..r-1) \leq \nu \leq z(l+1..n)$.

This is easily established by $\nu := (z(m)+z(n))/2$. The condition $z(m..r-1) \leq z(l)$ may then be strengthened to $\nu \leq z(l)$, and so on.

Although all three guards are strengthened, they are jointly still weak enough for the establishment of R or for the proper termination of the alternative construct.

*) Remarkably, the former choice, which is the obvious and exact implementation of the "formally derived" algorithm, yields a better performance than the latter, (Van Emden 70). Thus once more there is evidence that quite formal developments may lead to practically satisfactory programs.

9. References

Dijkstra, E.W. , 1976: A Discipline of Programming, Prentice Hall Inc., Englewood Cliffs, N.J.

Van Emden, M., 1970: Increasing the efficiency of Quicksort, C. ACM 13 (1970), 563-567, 693-694.

Hehner, E.C.R., 1976: do considered od, A contribution to the programming calculus, University of Toronto, Revised January 1978.

Hoare, C.A.R., 1961: Algorithm 63 Partition, Algorithm 64 Quicksort, C. ACM 4, (1961) 321.

Wirth, N., 1976: Algorithms + Datastructures = Programs, Prentice Hall Inc., Englewood Cliffs, N.J.

*) NB we kregen dus (uit §2):

do r ≠ n+1 cand z(r) ≤ rightmin → r := r+1; "pas leftmax aan"
▯ m ≠ l cand leftmax ≤ z(l) → l := l-1; "pas rightmax aan"
▯ r ≤ l cand leftmax ≤ z(r) ≥ z(l) ≤ rightmin → swap; l, r := l-1, r+1; "pas aan"
od.

Van Emden versterkt de eerste twee guards om "aanpassingen" zo weinig mogelijk te hoeven doen:
(dwz: definitieve splits.waarde zo lang mogelijk vrij te laten)
immers omdat leftmax ≤ rightmin geldt
z(r) < leftmax ⇒ z(r) < rightmin
en zelfs ook: r < l, i.e. r ≠ n+1

~~Helaas~~ niet zo voor de tweede guard
~~bewijst de~~ blijven die guards samen niet tolerant genoeg ~~[illegible]~~:

do z(r) < leftmax → r := r+1
▯ rightmin < z(l) → l := l-1
▯ leftmax ≤ z(r) ≥ z(l) ≤ rightmin → swap; r:+1; l:-1; "pas aan"
od

heeft extra alternatief nodig: leftmax ≤ z(l) ≥ z(r) ≤ rightmin → r:+1; l:-1; "pas aan".