TECHNISCHE HOGESCHOOL TWENTE
ONDERAFDELING DER TOEGEPASTE WISKUNDE

DISCIPLINED CONSTRUCTIONS OF PARTITION

Maarten M. Fokkings
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
2. The program specification
3. Developments maintaining \( l < r \)
4. Developments establishing \( l < r \)
5. "Wirth's trick" as an optimizing program transformation
6. Optimization by restricting nondeterminacy
7. A general treatment of some (tricky?) transformations
8. Further implementation
9. References
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 repetitive 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 \( z \log_{2} m n \leq \log_{2} n \), and integer variables \( l, r \), it is the purpose of the program to permute the array \( z(m..n) \) and partition it into a lefthalf \( z(m..l) \) of small elements, a righthalf \( z(r..n) \) 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 
\( z: \text{swap} \) 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: \( z: \text{swap} \) is the only value changing operation allowed on \( z \).
(The remainder of) the relation to be established is easily formalized as 
follows.

R: \( m-1 \leq l < n \) and \( m < r \leq n+1 \) and \( l < r \) and 
\( \forall v. z(m..l \leq v \leq z(r..n)) \) and \( v = z(l+1..r-1) \).

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 \( v = z(l+1..r-1) \). 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 subtract one from the difference 
in order to obtain lower-bound zero. Thus we obtain the invariant relation 
- P1 and variant function T1:

\[
P1: m-1 \leq l < n \quad \text{and} \quad m < r \leq n+1 \quad \text{and} \quad l < r \quad \text{and} \quad z(m..l) \leq z(r..n),
\]

T2: \( r - l - 1 \).

We will now try to develop the repetition do "maint P1 dcr T1" od (this 
abbreviates do "maintain P1 and decrease T1" od).

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 \( wdec(l := l+1, T1) \).

\[
wp(l := l+1, P1) = 
\]
\[
= p1[l+1+l+1]
\]
\[
= m-1 \leq l+1 < n \quad \text{and} \quad m < r \leq n+1 \quad \text{and} \quad l+1 < r \quad \text{and} \quad z(m..l+1) \leq z(r..n)
\]
\[
= l+1 \neq l \quad \text{and} \quad l+1 \neq r \quad \text{and} \quad z(l+1) \leq z(r..n), \quad \text{provided} \quad P1 \quad \text{holds}
\]
\[
= n \neq l+1 \neq r \quad \text{and} \quad z(l+1) \leq z(r..n), \quad \text{provided} \quad P1 \quad \text{holds}.
\]

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

\[
wdec(l := l+1, T1) =
\]
Thus we find as a suitable "step, decreasing \( T_1 \) while maintaining \( P_1 \) , \( A_1 \) and analogously \( A_2 \) :

\[
A_1: \text{not} l+1 \not\equiv \text{cand} \, z(l+1) \leq z(r..n) \rightarrow l := l+1 \ ,
\]

\[
A_2: \text{not} r-1 \not\equiv \text{cand} \, z(m..l) \leq z(r-1) \rightarrow r := r-1 .
\]

Note that we have replaced and by \( \text{cand} \) in order to make the evaluation of the second term well defined.

Not surprisingly, the alternatives \( A_1 \) and \( A_2 \) are not sufficient; do \( A_1 \) \& \( A_2 \) od does not establish \( R \) . Therefore we look for further alternatives decreasing \( T_1 \) while maintaining \( P_1 \) . It is left to the reader to imagine that \( z: \text{swap}(l+1, r-1) \); \( l, r := l+1, r-1 \) may do when neither of the guards of \( A_1 \) and \( A_2 \) holds. Formal calculation shows:

\[
\begin{align*}
wp(z: \text{swap}(l+1, r-1)); \ l, r := l+1, r-1 , \ P_1) &= \\
= wp(z: \text{swap}(l+1, r-1), \ wp(l, r := l+1, r-1 , \ P_1)) &= \\
= wp(z: \text{swap}(l+1, r-1), \ P_1[l, r + l+1, r-1]) &= \\
= P_1[l, r + l+1, r-1][z + z']
\end{align*}
\]

here and in what follows \( z' \) is defined by:

\[
\begin{align*}
z'(i) &= \text{if } i = l+1 \rightarrow z(x-1) \text{ and } i = r-1 \rightarrow z(l+1) \text{ and } 1 \not\equiv \text{not} i \not\equiv \text{not} r-1 \rightarrow z(i) \text{ fi} \\
&= m-1 \leq l+1 \text{ and } m = r-1 \text{ and } l+1 \leq z'(r-1..n) \\
&= r \not\equiv \text{not} 1 \not\equiv \text{not} l \text{ (provided } P_1 \text{ holds)} \text{ and } z'(m..l) \leq z'(r-1..n) \\
&= 2 \not\equiv \text{not} 1 \not\equiv \text{not} (z'(m..l), z'(l+1)) \leq ((z'(r-1), z'(r..n)) \text{ provided } P_1 \text{ holds.}
\end{align*}
\]

Secondly,

\[
\begin{align*}
\text{wdec}(z: \text{swap}(l+1, r-1)); \ l, r := l+1, r-1 , \ T_1) &= \\
= \text{tmin} \leq T_1 \text{-1 where } \text{tmin} = \text{min} \ t_0. \ wp(\text{not} \ T_1 \leq t_0) &= \\
= \text{tmin} = \text{min} \ t_0. \ (T_1 \leq t_0)[l, r + l+1, r-1][z + z'] &= \\
= \text{tmin} = \text{min} \ t_0. \ (r-1) - (l+1) - 1 \leq t_0 &= \\
= \text{tmin} = r-3 &= \\
= r-3 \leq r-2 &= \\
= \text{true} .
\end{align*}
\]

Hence a suitable alternative is:

\[
A_3: 2 \not\equiv \text{not} 1 \not\equiv \text{not} (z(m..l) \leq z(l+1) \leq z(r-1) \leq z(r..n)) \rightarrow \\
\text{z: \text{swap}(l+1, r-1); l, r := l+1, r-1} .
\]
Unfortunately we are still not through. With the initialization
\[ l,r := m-1, n+1 \] the repetition do A1 □ A2 □ A3 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 \( z(l+1) \leq z(r..n) \) 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 emptiness of \( m..l \) and \( r..m \) may easily be avoided by the following initialization:

\[
\begin{align*}
\text{if } z(m) \leq z(n) & \Rightarrow \text{ skip } \Box z(m) \geq z(n) + z: \text{swap}(m,n) \ 	ext{fi;}
\end{align*}
\]

\[ l,r := m,n \] (note that \( l<r \) requires \( m<n \)).

Indeed, \( m \leq l \) and \( r \leq n \) 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:

\[ P2: P1 \text{ and } m \leq l \text{ and } r \leq n, \text{ or simplified,} \]

\[ m \leq l < r \leq n \text{ and } z(m..l) \leq z(r..n). \]

\[ \text{S1: do A4 □ A5 □ A6 od}, \]

\[ A4: l+1 \neq r \text{ cand } z(l+1) \leq z(r..u) + l := l+1; \]

\[ A5: l \neq r-1 \text{ cand } z(m..l) \leq z(r-1) + r := r-1; \]

\[ A6: 2 \neq r-1 \text{ cand } z(m..l) \leq z(l+1) \leq z(r-1) \leq z(r..n) + z: \text{swap}(l+1,r-1); i,r := l+1,r-1. \]

The above initialization and repetition establish R.

Proof. On account of \( m \leq l \) and \( r \leq n \)

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

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

\[ (2a,b) \ z(l+1) \leq z(r..n), \text{ or on account of } P2 \text{ and } (1), \ z(m..l) \leq z(l+1), \]

\[ (3a,b) \ z(m..l) \leq z(r..n), \text{ or on account of } P2 \text{ and } (1), \ z(r-1) < z(r..n), \]

From A6 there arise four possibilities:

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

or: \( z(m..l) \leq z(l+1) \) but this contradicts (2b),

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

or: \( z(r-1) < z(r..n) \) but this contradicts (2b).

Hence all possibilities lead to a contradiction. Therefore \( l+1 \neq r-1 \) 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 \( m < n \), 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 \( \text{do "mnt P2 dcr} 
T1\text" od \). We will now take another strategy. Note that \( P2 \) and \( r - l = 1 \)
implies \( R \). Thus we try to develop the following repetition:
\[ \text{do } r \neq l \neq 1 + \text{"gvn } r \neq l \neq 1 \text{ mnt P2 dcr } T1\text" \text{ od }. \]

Formal calculations show:
\[
\begin{align*}
wp(l := l + 1, P2) &= z(l + 1) \leq z(r..n) & \text{provided } P2 \text{ and } r \neq l \neq 1 \text{ holds}, \\
wp(r := r - 1, P2) &= z(m..l) \leq z(r - 1) & \text{provided } P2 \text{ and } r \neq l \neq 1 \text{ holds}, \\
wp(z : \text{swap}(l + 1, r - 1); l, r := l + 1, r - 1, P2) &= \\
& l + 1 \neq r - 1 \text{ and } z(m..l) \leq z(l + 1) \leq z(r - 1) \leq z(r..n) & \text{provided } P2 \text{ and } r \neq l \neq 1 \text{ holds},
\end{align*}
\]
The \( \text{wdec} \) of any of these statements with respect to \( T1 \) equals 
\( \text{true} \), and moreover if \( r \neq l \neq 1 \) then certainly one of the conditions 
will hold. Thus the repetition may read
\[
\begin{align*}
S2: \text{do } r \neq l \neq 1 + \\
& \text{if } z(l + 1) \leq z(r..n) + l := l + 1 \\
& \quad \Box z(m..l) \leq z(r - 1) + r := r - 1 \\
& \quad \Box l + 1 \neq r - 1 \text{ and } z(m..l) \leq z(l + 1) \leq z(r - 1) \leq z(r..n) + \\
& \quad \text{z : swap}(l + 1, r - 1); l, r := l + 1, r - 1 \\
& \quad \text{fi} \\
& \text{od}.
\end{align*}
\]

Note that the proof of the establishment of \( R \) is now replaced by 
esentially 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 \( l+1 \neq r-1 \). Indeed, the disallowed initial state satisfying \( m=n \) might lead to proper termination with unreliable results (if e.g. \( z\cdot \text{low} \leq m < n < z\cdot \text{hib} \)) in \( S_2 \), 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 \( r \leq z \) (which unfortunately detracts from robustness) and the repetition is continued with

\[
\begin{align*}
\text{if} & \quad l=r \quad \text{or} \quad l=r+1 \\
\text{then} & \quad \text{skip}
\end{align*}
\]

The easily verifiable invariant relation then reads

\[ P_2 \text{ or } (z(m..l) \leq z(r..n)) \]

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

4. Developments establishing \( l \leq r \)

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

\[
\begin{align*}
& l \leq r \text{ and } \text{Ev. } z(m..l) \leq z(r..n) \text{ and } v=z(l+1..r-1) \\
= & \quad l \leq r \text{ and } z(m..l) \leq z(r..n) \text{ and } z(m..l) \leq z(l+1..r-1)=z(l+1..r-1) \leq z(r..n) \\
= & \quad l \leq r \text{ and } z(m..l-1) \leq z(l+1..n) \\
\end{align*}
\]

Thus the full relation to be established now becomes

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

The above relation, although equivalent to the original \( R \), suggests a quite different invariant: drop the term \( l \leq r \), yielding

\[ P_3: m-1 \leq i < n \text{ and } m \leq r \leq n+1 \text{ and } z(m..l-1) \leq z(l+1..n) \]

This relation is easily established by

\[
\begin{align*}
\text{if} & \quad z(m) \leq z(n) \quad \text{skip} \quad z(m) \geq z(n) \quad z:=\text{swap}(m,n) \quad \text{fi;} \\
& \quad r, l := m+1, n-1
\end{align*}
\]

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 $T2: n = n + 1 - r$.

Similarly to the previous section, we first try to develop $\text{do } "\text{mnt P3 dcr T2}" \text{ od }$.

Obvious candidates for a decrease of $T2$ are the statements $z := z + 1$ and $z := z + 1$. Formal calculation shows

\[wp(l := l - 1, P3) = m - 1 \neq l \text{ and } z(m \cdot r - 1) \leq z(l)\]

\[wp(r := r + 1, P3) = r \neq n + 1 \text{ and } z(r) \leq z(l + 1 \cdot n)\]

\[wp(z := swap(r, l); r, l := r + 1, l - 1, P3) =
\begin{align*}
\text{r} \neq l \text{ and } z(m \cdot r - 1) &\leq z(r) \leq z(l)\leq z(l + 1 \cdot n) \text{ or} \\
\text{m} - 1 \neq l \neq n + 1 \text{ and } z(r) &= z(l)
\end{align*}

provided $P3$ holds.

For each of these statements $S$, $\text{wdec}(S, T2) = \text{true}$, thus we arrive at the following alternatives.

\[A7: m - 1 \neq l \text{ cand } z(m \cdot r - 1) \leq z(l) \Rightarrow l := l - 1\]

\[A8: r \neq n + 1 \text{ cand } z(r) \leq z(l + 1 \cdot n) \Rightarrow r := r + 1\]

\[A9: r \neq l \text{ cand } z(m \cdot r - 1) \leq z(r) \leq z(l) \leq z(l + 1 \cdot n) +
\text{z := swap(r, l); r, l := r + 1, l - 1}\]

(Rather arbitrarily we have omitted the disjunct $m - 1 \neq l \neq n + 1 \text{ cand } z(l) = z(r)$ 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

$S3: \text{do } A7 \{ A8 \{ A9 \text{ od }\}$

establishes $l < r$ (check!) hence $R$.

***

Above we have developed the scheme $\text{do } "\text{mnt P3 dcr T2}" \text{ od }$. We will now describe the development of another scheme. First note that $P3$ and $l < r$ implies $R$. Thus choose the scheme

$\text{do } l < r + "\text{gvn r < l mnt P3 dcr T2}" \text{ od }$.

Formal calculation shows that

\[wp(l := l - 1, P3) = z(m \cdot r - 1) \leq z(l)\]

\[wp(r := r + 1, P3) = z(r) \leq z(l + 1 \cdot n)\]

\[wp(z := \text{swap}(r, l); r, l := r + 1, l - 1, P3) =
\begin{align*}
\text{z(m \cdot r - 1) &\leq z(r) \leq z(l) \leq z(l + 1 \cdot n) provided P3 and r < l holds.}
\end{align*}

Further, $P3$ and $r < l$ implies that at least one of these conditions holds.
Thus we obtain the repetition

\[ S4: \text{do } r \leq l \rightarrow \]

\[
\begin{align*}
& \text{if } z(m..r-1) \leq z(l) \rightarrow l := l-1 \\
& \quad \text{z}(r) \leq z(l+1..n) \rightarrow r := r+1 \\
& \quad \text{z}(m..r-1) \leq z(r) \leq z(l) \leq z(l+1..n) \rightarrow z: \text{swap}(r, l); r, l := r+1, l-1 \\
& \text{fi}
\end{align*}
\]

\[ \text{od}. \]

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:

\[ S5: \text{"mnt P3 est R"}:
\]

\[
\begin{align*}
& \text{if } l < r \rightarrow \text{skip} \\
& \quad \text{m} - 1 \neq l \text{ cand } z(m..r-1) \leq z(l) \rightarrow l := l-1, \text{ "mnt P3 est R"} \\
& \quad r \neq n+1 \text{ cand } z(r) \leq z(l+1..n) \rightarrow r := r+1; \text{ "mnt P3 est R"} \\
& \quad r \leq l \text{ cand } z(m..r-1) \geq z(r) \geq z(l) \leq z(l+1..n) \rightarrow \\
& \quad \quad z: \text{swap}(r, l); r, l := r+1, l-1; \text{ "mnt P3 est R"}
\end{align*}
\]

\[ \text{fi}. \]

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 $z < r$ and to those establishing $z < r$.

Recall from section 3 repetition $S_1$ maintaining $P_2$.

$P_2$: $m \leq z < r \leq n$ and $z(m..l) \leq z(r..n)$,

$S_1$: do $A_4 \parr A_5 \parr A_6$ od,

$A_4$: $l+1 \not= r$ cand $z(l+1) \leq z(r..n)$ + $z := l+1$,

$A_5$: $l \not= r-1$ cand $z(m..l) \leq z(r-1)$ + $r := r-1$,

$A_6$: $l \not= r \not= 2$ cand $z(m..l) \leq z(l+1) \leq z(r-1) \leq z(r..n)$ + $z := \text{swap}(l+1, r-1); l, r := l+1, r-1$.

Now strengthen the guard of $A_4$ 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 $A_5$ (independently of its guard) and $A_6$, and it is also initially true with $z, r := m, n$ (because $m < n$ is already assumed). Analogously the guard of $A_5$ may be strengthened. Thus we obtain the alternatives

$A_4'$: $z(l+1) < z(r..n)$ + $z := l+1$,

$A_5'$: $z(m..l) < z(r-1)$ + $r := r-1$.

Although two guards have been strengthened, repetition $S_1''$: do $A_4' \parr A_5' \parr A_6$ od still establishes the emptiness of $z+1..r-1$ hence $R$.

Similarly we may change $S_3$ which maintains $P_3$.

$P_3$: $m \leq z < n$ and $m \leq z+1$ and $z(m..l+1) \leq z(l+1..n)$,

$S_3$: do $A_7 \parr A_8 \parr A_9$ od,

$A_7$: $m-1 \not= l$ cand $z(m..l-1) \leq z(l)$ + $l := l-1$,

$A_8$: $n \not= r+1$ cand $z(r) \leq z(l+1..n)$ + $r := r+1$,
A9: \( r \leq l \) and \( z(m..r-1) \leq z(l) > z(l+1..n) \) +  
\[ z: \text{swap}(r, l); r, l := r + 1, l - 1 \]  
Now strengthen the guard of \( A7 \) into \( z(m..r-1) < z(l) \). Then \( m \leq l \) 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 \( m < n \). Analogously 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  
\[ S3': \text{do } A7' \text{ od } A8' \text{ od } A9 \text{ od} \]  
still establishes \( l < r \) hence \( R \).  

6. Optimization by restricting nondeterminacy  
For simplicity we only consider repetition \( S3 \). Recall  
\[ S3: \text{do } A7 \text{ od } A8 \text{ od } A9 \text{ od} \]  
\[ A7: m \neq l \]  
\[ \text{cand } z(m..r-1) \leq z(l) + l := l + 1 \]  
\[ A8: r \neq n + 1 \]  
\[ \text{cand } z(r) \leq z(l+1..n) + r := r + 1 \]  
\[ A9: r \leq l \]  
\[ \text{cand } z(m..r-1) < z(r) \leq z(l) > z(l+1..n) + z: \text{swap}(r, l); r, l := r + 1, l - 1 \]  
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 superfluous.  

First, note that \( \text{not } (A7.\text{guard or A8.\text{guard}) and r \leq l} \) implies \( A9.\text{guard} \). Therefore we "group together the potential steps over \( A7 \) and \( A8 \), so that thereafter in (a single, potential execution of) \( A9 \) the term \( z(m..r-1) \leq z \l s.. \) is superfluous. This transformation yields:  
\[ \text{do } A7.\text{guard or A8.\text{guard or A9.\text{guard}} + do } A7 \text{ od A8 od} \]  
\[ \text{if } r \leq l + z: \text{swap}(r, l); r, l := r + 1, l - 1 \text{ od } l < r \text{ skip \text{fi} do} \]  
Second, note that the main guard of the above repetition is at least as weak as \( r \leq l \), whereas the latter is already sufficient for the establishment of \( R \). Thus strengthen the main guard into \( r \leq l \).  
Third, the relation \( \text{not } A7.\text{guard} \) is invariant over \( A8 \), and \( \text{not } A8.\text{guard} \) is invariant over \( A7 \). Therefore the inner repetition may be particularized into \( \text{do } A7 \text{ od; do } A8 \text{ od} \).  
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": do r≠l→
   do A4 od; do A5 od;
   if 2≠r≠l → 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 "mmt 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
developing 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 sequen-
tial 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
\[
\text{if } r\leq l + z: \text{swap}(r, l); \quad r, l := r + 1, l - 1 \quad \square l < r + \text{skip fi}
\]
by
\[
z: \text{swap}(r, l); \quad r, l := r + 1, l - 1
\]
provided afterwards we undo a possible unwanted swap. Thus we obtain
S3*: \[\text{do } r \geq l +
\]
\[
\text{do } A7 \text{ od}; \quad \text{do } A8 \text{ od};
\]
\[
z: \text{swap}(r, l); \quad r, l := r + 1, l - 1
\]
\[
\text{od;}
\]
\[
\text{if } r - 1 \leq l + 1 \rightarrow \text{skip } \square l + 1 < r - 1 \rightarrow l, r := l + 1, r - 1; \quad z: \text{swap}(l, r) \text{ fi}
\]
The invariant relation then reads
Q: P3 or
\[
l + 1 < r - 1 \text{ and } (z(m..l - 1), z(r), z(l + 1..r - 1)) \leq (z(l + 1..r - 1), z(l), z(r + 1..n)).
\]
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
\[
i := 0;
\]
\[
\text{while } i \neq n
\]
\[
\text{do if } i = 0 \text{ then sum} := 0;
\]
\[
i := i + 1; \quad \text{sum} := \text{sum} + t(i);
\]
\[
\text{if } i = n \text{ then print } (\text{sum}) \quad \text{od .}
\]
Similarly, I find it not elegant and not efficient that in \( S_3', S_3'', S_1', \) and \( S_1'' \) 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 \( A \implies wp(B,C) \).  

Theorem. Consider a repetitive construct of the form  
\[ S: \text{do } \{P\} \text{ C } \rightarrow SL0; \{X\} \text{ if } B1 \rightarrow SL1 \[ \square B2 \rightarrow SL2 \{P\} \text{ od } \]  
with the following properties.  
First, \( \text{ P } \) is an invariant relation and \( \text{ X } \) holds at the point indicated:  
1. \( \{P \text{ and } C\} \text{ SLO } \{X \text{ and } (B1 \text{ or } B2)\} \),  
2. \( \{X \text{ and } B1\} \text{ SL1 } \{P\} \),  
3. \( \{X \text{ and } B2\} \text{ SL2 } \{P\} \).  
Second, if the alternative construct is executed when \( B2 \) holds, then the repetition terminates, even if the whole alternative construct is replaced by \( \text{ SL1 } \):  
4. \( \{X \text{ and } B2\} \text{ SL1 } \{\neg C\} \),  
5. \( \{X \text{ and } B2\} \text{ SL2 } \{\neg C\} \).  
Third, if the alternative construct has been replaced by \( \text{ SL1 } \), it is decidable afterwards whether \( \text{ SL1 } \) has been executed rightly or wrongly, by testing mutually exclusive conditions \( B1' \) and \( B2' \):  
6. \( \{X \text{ and } B1\} \text{ SL1 } \{C \text{ or } B1'\} \),  
7. \( \{X \text{ and } B2\} \text{ SL1 } \{B2'\} \),  
8. \( B1' \text{ and } B2' = \text{ false} \).  
And let \( \text{ SL1}^{-1} \) be any statement which undoes the effect of the "wrongly" execution of \( \text{ SL1 } \) in a state \( X \text{ and } B2 \):  
9. \( \{X \text{ and } B2\} \text{ SL1; SL1}^{-1} \{X \text{ and } B2\} \).  
And assume finally  
10. the initial establishment of \( \text{ P } \) also establishes \( C \text{ or } B1' \).  
Then the postassertion \( P \text{ and } \neg C \) of \( S \) is as well established by the optimized program
$S': \mathsf{do} \ C \to \mathsf{SL0}; \ \mathsf{SL1} \ \mathsf{od}; \ \mathsf{if} \ B1' \to \mathsf{skip} \ [ \ B2' \to \mathsf{SL1}^{-1}; \ \mathsf{SL2} \ \mathsf{fi} \ .$

Proof. Let $Q$ be defined by

$$Q: (P \mathsf{and} (C \mathsf{or} B1')) \mathsf{or} (QQ \mathsf{and} C \mathsf{and} B2')$$

where $QQ$ is any relation which satisfies

$$(11) \ \{X \mathsf{and} B2; \ SL1 \ \{QQ\} \ SL1^{-1} \ \{X \mathsf{and} B2\} \ .$$

(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 $(B1 \mathsf{or} B2)$ holds just before $SL1$. By (2) and (6) we find

$$(X \mathsf{and} B1) \ SL1 \ \{P \mathsf{and} (C \mathsf{or} B1')\} \ , \ \text{and by (9), (5) and (7) we find}$$

$$(X \mathsf{and} B2) \ SL1 \ \{QQ \mathsf{and} C \mathsf{and} B2'\} \ . \ \text{Hence $Q$ is reestablished upon exit from the repeatable statement.})$$

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

$S3'': \mathsf{do} \ r \leq z \to$

$$\mathsf{do} \ A7 \ \mathsf{od}; \ \mathsf{do} \ A8 \ \mathsf{od}; \ $$

$$\mathsf{if} \ r \leq z + z:swap(r,z); \ r,z:=r+1,z-1 \ \mathsf{if} \ z<r \to \mathsf{skip} \ \mathsf{fi} \ \mathsf{od} \ .$$

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

$S3'' = \mathsf{do} \ C \to SL0; \ \mathsf{if} \ B1 \to SL1 \ [ \ B2 \to SL2 \ \mathsf{fi} \ \mathsf{od} \ .$

Note that $X$ is the invariant relation $P3$. We choose

$B1'': r+1 \leq z-1,$

$B2'': z-1 \leq r+1,$

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

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

$S3*: \mathsf{do} \ C \to SL0; \ SL1 \ \mathsf{od}; \ \mathsf{if} \ B1' \to \mathsf{skip} \ [ \ B2' \to SL1^{-1}; \ \mathsf{SL2} \ \mathsf{fi} \ \mathsf{od} \ .$

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 \( P, C, B_1' \) and \( B_2' \), but not \( QQ \). Indeed, condition (9) can be proved without implicitly deriving \( QQ \), as follows. First, any predicate \( P \), on which \( SL_1 \) will properly terminate, is invariant over \( SL_1 SL_1^{-1} \) (which may be proved by textual manipulation). Second, from (3) it follows that \( SL_1 \) will properly terminate on \( X \) and \( B_2 \). (End of remark.)

The intuitively similar transformation of remark 2 in section 3 can’t be proved by the above theorem. True, program \( S_2 \) can be brought into the required form by trivial textual manipulations, but no mutually exclusive relations \( B_1' \) and \( B_2' \) can be found, and quite essentially, \( z(m..l) \leq z(l+1) \geq z(r-1) \geq z(r..n) \) 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 \( \text{leftmax} \) and \( \text{rightmax} \) which invariently satisfy, e.g.,

\[
\text{leftmax} = \max z(m..r-1)
\]

(or \( \text{leftmax} = \max(z(m..r-1),-\infty) \) if \( m..r-1 \) can be empty).

The condition \( z(m..r-1) \leq z(l) \) may then be represented by \( \text{leftmax} \leq z(l) \) 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 \( v \), invariently satisfying, e.g.,

\[
z(m..r-1) \leq v \leq z(l+1..n)
\]

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


