Program inversion.

Let the integer array $p(0..M-1)$ be such that the sequence $p(0)$, $p(1),\ldots, p(M-1)$ represents a permutation of the numbers from 0 through $M-1$ and let the integer array $y(0..M-1)$ be such that $(\underline{A}\ i: 0 \leq i < M: 0 \leq y(i) \leq i)$. Under those constraints we are interested in the relation

$$(\underline{A}\ i: 0 \leq i < M: y(i) = (\underline{N}\ j: 0 \leq j < i: p(j) < p(i))\ ) \qquad (1)$$

(Legenda: "$(\underline{N}\ j: 0 \leq j < i: p(j) < p(i))$" should be read as "the number of mutually different values j in the range $0 \leq j < i$, such that $p(j) < p(i)$".)

We can now consider the two --solvable-- problems

A) Given p , assign to y a value such that (1) is satisfied.
B) Given y , assign to p a value such that (1) is satisfied.

Because we want to consider programs the execution of which may modify the given array, we rephrase:

A) Given p , assign to y a value such that (1) holds between the initial value of p and the final value of y .
B) Given y , assign to p a value such that (1) holds between the initial value of y and the final value of p .

If A transforms p into a (standard) value which is its initial value in B , and if B transforms y into a (standard) value which is its initial value in A , then transformations A and B are <u>inverse</u> transformations on the pair (p,y). We are interested in these inverse transformations because in general problem A is regarded as easier than B : we have solved problem B as soon as we have for A a reversible solution!

Our first effort.

Let the standard value for p be such that $(\underline{A}\ i: 0 \leq i < M: p(i) = i)$. From (1) we immediately deduce that a permutation of the values $p(0),\ldots,$ $p(k-1)$ does not affect the values of $y(i)$ for $i \geq k$. This suggests the computation of the values $y(k)$ in the order of increasing k , each time combining the computation of $y(k)$ with a permutation of $p(0),\ldots,$ $p(k)$. Because the final value of p should be sorted, we are led most naturally to a bubble sort:

```
k:= 0; {p(0),...,p(k-1)  is ordered}
do k ≠ M → "make  p(0),.., p(k) ordered";
           k:= k + 1 {p(0),..., p(k-1)  is ordered}
od
```

The standard program for the bubble sort is

```
k:= 0;
do k ≠ M → j:= k;
           do j > 0 cand p(j-1) > p(j) → p:swap(j-1,j);
                                         j:= j - 1
           od {here  j = the value  y(k)  should get};
           k:= k + 1
od {A i: 0 ≤ i < M : p(i) = i}
```

We initialize via $y:=(0)$ the array variable y as the empty array with $y.lob = 0$, each time extending it with a new value as soon as that has been computed. Because $k = y.dom$ would be an invariant, the variable k can be eliminated.

Program A1:

```
y:=(0); {y.dom = 0}
do y.dom ≠ M → j:= y.dom {this is an initialization}; {j = y.dom}
               do j > 0 cand p(j-1) > p(j) → p:swap(j-1,j);
                                             j:= j - 1 {j < y.dom}
               od; y:hiext(j) {j's value is no longer relevant} {y.dom > 0}
od {A i: 0 ≤ i < M: p(i) = i}
```

Inverting it we construct

Program B1:

```
p:=(0); do p.dom ≠ M → p:hiext(p.dom) od; {A i: 0 ≤ i < M: p(i) = i }
do y.dom ≠ 0 → j,y:hipop {this is an initialization of  j };
               do j ≠ y.dom → j:= j + 1; p:swap(j-1,j) od
                  {j's value is no longer relevant}
od
```

This inversion was easy because the post-condition of each repeatable statement implies the negation of the stated precondition of the repetitive construct as a whole; furthermore we have used that y:hiext(j) and j,y:hipop are each other's inverse, that j:= j + 1 and j:= j - 1 are each others inverse, and that p:swap(j-1,j) is its own inverse.

We leave to the reader the insertion of provable assertions in program B1 that would justify the derivation of A1 from B1 by inversion.

<u>Our second effort</u>.

We can also compute the values $y(k)$ in the order of decreasing k. (Here it is as if our standard value of p is the empty array with $p.lob = 0$ and the standard value of y is the empty array with $y.hib = M - 1$.) We make three observations:

1) As soon as the $y(i)$ for $i \geq k$ have been computed, the $p(i)$ for $i \geq k$ no longer matter, i.e. we can work with a single array, $v(0..M-1)$ say, where in A/B, in relation (1) p refers to the initial/final value of v, and y refers to the final/initial value of v.

2) Denoting with $Q(k)$: "the sequence $p(0), p(1), ..., p(k)$ represents a permutation of the numbers $0, ..., k$" we can write $Q(k) \Rightarrow y(k) = p(k)$.

3) Decreasing in the range $0 \leq i < k$ all $p(i)$ such that $p(i) > p(k)$ by 1 leaves all $y(i)$ with $0 \leq i < k$ unaffected.

These observations lead to the following program (in which we can view the elements $v(i)$ with $i < k$ as the corresponding elements of (a changing) p and the $v(i)$ with $i \geq k$ as the corresponding elements of a growing y.)

```
k:= M; {k = M and Q(k-1) and v = p}
do k ≠ 0 → k:= k - 1; {Q(k)}
        i:= 0; do i ≠ k → if v(i) > v(k) → v:(i)= v(i) - 1 {v(i) ≥ v(k)}
                           ▯ v(i) < v(k) → skip {v(i) < v(k)}
                          fi; i:= i + 1
                   od {i = k and Q(k-1)}
od {k = 0 and v = y}
```

In the alternative construct the postconditions have been added in order to ease the inversion:

```
Program  B2:
k:= 0 {v = y};
do k ≠ M → i:= k;
           do i ≠ 0 → i:= i - 1;
                      if v(i) ≥ v(k) → v:(i)= v(i) + 1
                       ▯ v(i) < v(k) → skip
                      fi
           od {i = 0};
           k:= k + 1
od {k = M and v = p}.
```

*   *   *

The problems A and B I had invented for examination purposes. After the students had handed in their work, it was W.H.J.Feijen who suggested that it would be nice to derive the one program from the other via inversion. Because in this case we have a deterministic program in which no information is destroyed, the inversion is a straightforward process. What remains of these techniques in the general situation remains to be seen. Is it possible to show that a program with nondeterministic elements leads to a unique answer because in its inverse no information is destroyed? Who knows.... In the meantime I have derived a program --B2 to be precise-- that was new for me.

Plataanstraat 5
5671 AL NUENEN
The Netherlands

prof.dr.Edsger W.Dijkstra
BURROUGHS Research Fellow