The problem of the table of cubes

For $0 \leq N$, we consider the following little program:

$$n := 0$$
$$;\ \underline{\text{do}}\ n \neq N \rightarrow \text{print}(n^3);\ n := n+1\ \underline{\text{od}}\quad .$$

It does precisely what we want it to do, viz. print the first N cubes of the natural numbers. Unfortunately, it is of no use for our currently imagined computer installation, which has no facilities for exponentiation or multiplication, but can only cope with additive arithmetic operations. We therefore proceed a little further and eliminate the unwanted expression n^3.

The way to eliminate it is standard: we replace "print (n^3)" with

$$\{x = n^3\}\ \text{print}(x) \ ,$$

and the obligation then left is to ascertain that the precondition of the print-statement satisfies $x = n^3$ indeed. Since that precondition at the same time is a precondition of the entire repeatable statement (i.e. the step of the repetition), we ensure it by adopting

P0: $\quad x = n^3$

as an invariant of the repetitive construct.

Thus we obtain for our program

$$
\begin{array}{l}
\quad n, x := 0, 0 \\
;\ \underline{\text{do}}\ n \neq N \ \rightarrow \\
\qquad \{P0\} \quad \text{print } (x) \\
\qquad ;\ \{P0\} \quad n, x := n+1, \ \ldots \\
\quad \underline{\text{od}} \ .
\end{array}
$$

The obvious way to maintain $P0$ is to substitute $(n+1)^3$ for the dots, which is correct regardless even of the precondition of the assignment. But we cannot leave it at that, since the expression $(n+1)^3$ is at least as unwanted as the original expression n^3 which we try to eliminate. Using the precondition, however, we observe

$$
\begin{array}{ll}
 & (n+1)^3 \\
= & \quad \{\text{arithmetic}\} \\
 & n^3 + 3 \times n^2 + 3 \times n + 1 \\
= & \quad \{\text{by precondition } P0\} \\
 & x + 3 \times n^2 + 3 \times n + 1 \ ,
\end{array}
$$

and this is all we can say. But substituting the latter expression for the dots is still no good, although a little better, because the highest exponent of n has decreased from 3 to 2.

Now we have recourse to the same standard technique as before. The derived assignment

$$n, x := n+1, \ x + 3 \times n^2 + 3 \times n + 1$$

is replaced with

$$\{P1\} \quad n, x := n+1, x+y \quad ,$$

under simultaneous adoption of

P1: $\quad y = 3*n^2 + 3*n + 1$

as an additional invariant of the repetitive construct. Thus, we obtain as a next version of our program — omitting the print-statement —

```
   n, x, y := 0, 0, 1
 ; do n ≠ N →
       {P0}{P1}
         n, x, y := n+1, x+y, ...
   od.
```

<u>Remark</u> For P1 we could also have chosen something like $y = 3*n^2 + 3*n$ or $y = n^2$. This would do too in that it would eliminate unwanted sub-expressions as well. The resulting assignment to x, however, would have been a little more complicated. (<u>End</u> of Remark.)

For maintaining P1. we proceed in a way that is completely analogous to our dealing with P0. For the obvious expression to be substituted for the dots, we observe

$$3*(n+1)^2 + 3*(n+1) + 1$$
$$= \quad \{\text{arithmetic}\}$$

$$
\begin{array}{ll}
 & 3*n^2 + 3*n + 1 \quad + \quad 6*n + 6 \\
= & \quad \{\text{by precondition } P1\} \\
 & y + \quad 6*n \quad + 6 \\
= & \quad \{\text{by adopting } P2\text{, given below}\} \\
 & y + z \text{ ,}
\end{array}
$$

where

$$P2: \qquad z = 6*n + 6$$

is yet an additional invariant.

For the benefit of maintaining $P2$, we finally observe

$$
\begin{array}{ll}
 & 6*(n+1) \quad + \quad 6 \\
= & \quad \{\text{arithmetic}\} \\
 & 6*n + 6 \quad + \quad 6 \\
= & \quad \{\text{by precondition } P2\} \\
 & z + 6 \text{ ,}
\end{array}
$$

and our ultimate program thus becomes

```
   n, x, y, z := 0, 0, 1, 6
   {inv. P0 ∧ P1 ∧ P2}
 ; do n ≠ N →
         n, x, y, z := n+1, x+y, y+z, z+6
   od
```

* * *

I have known this program and its derivation for almost a decade now, but I never recorded it on paper. (Well once, in fact, as a handout for a group of Finnish students visiting our university.) The reason for "hiding" it was that it was deemed so simple, that anybody could be trusted to find the above solution for himself. Quod non. Professionals that were interviewed about the problem, would, in general, immediately focus on a "difference scheme", which is not amazing because many of them still have some sort of background in numerical mathematics. Then, of course, the question would arise whether to focus on "forward" or "backward" differences. This, in combination with the absence in most programming languages of the multiple assignment then raises the problem in which order to put the simple assignments. It goes without saying that all these little inconveniences in solving the problem are the "fruits" of operational reasoning: they are never encountered when —as we did— the program is simply calculated from its functional specification.

The other group of people that have been interviewed about the problem consisted of computing science students, mostly at the sophomore or junior level, at both sides of the Atlantic. Many of them contented themselves with a very inefficient program, quadratic in N or worse, with no gut feeling for improvement. I found that alarming.

The more firm and technical moral of the above program derivation is that the introduction of a new variable always comes with the introduction of a new invariant, and this is good to know.

W.H.J. Feijen,
Sterksel, 30 April 1990.