Misra's weakest fair buffer

To begin with, we are interested in functions f of type $nat \to nat$ that are involutions — i.e. are their own inverses — and have no fixpoints, i.e.

(0) $\langle \forall n :: (f \circ f).n = n \wedge f.n \neq n \rangle$.

Because $f.p = q \Rightarrow f.q = p \wedge p \neq q$, function f partitions the natural numbers into pairs (p,q) of distinct values: in English, the "meaning" of f would be "the partner of" or "the other". Function f determines the partitioning into pairs; conversely, the partitioning determines f.

We say that a program generates f if it generates all pairs (p,q) satisfying $M.p.q$ with M given by

(1) $M.p.q \equiv f.p = q \wedge p < q$.

We shall consider separately generating the pairs in the order of increasing p and in the order of increasing q. We are particularly interested in the "weakest" programs for this job, i.e. programs so nondeterministic that they can generate any function f satisfying (0).

* * *

The program that generates the (p,q) pairs in the order of increasing p is straightforward. The first conjunct of the invariant, which relates a local variable to the amount of output generated is of an absolute standard structure

P0: $\langle\forall p,q :: M.p.q \wedge p<n \equiv$ "(p,q) has been generated"$\rangle$;

it is initially, when nothing has been generated, established by $n:=0$, and the desired answer will be generated by increasing n under invariance of P0.

In order to control that the pairs generated form a partitioning of the natural numbers, the second conjunct links a local variable — of type: Set of naturals — to the irrevocable commitments made beyond n, more precisely

P1: $\langle\forall q :: f.q<n \wedge n\leq q \equiv q\in S\rangle$;

it is initially established — with $n=0$ — by $S:=\emptyset$.

The following program does the job

```
|[ var n: nat = 0 ; var S: Set of nat = ∅
; do n ∈ S → S := S - {n} ; n := n+1
  [] n ∉ S → |[ var r: nat
              ; solve r: n < r ∧ r ∉ S
              ; generate (n, r)
              ; S := S + {r} ; n := n+1
              ]|
  od
]|        .
```

The algorithm's nondeterminacy is concentrated in the instruction to solve

$$r:\ n < r \ \wedge\ r \notin S \quad .$$

Since S is finite, constructing a solution is no problem; for the same reason the instruction to solve it is a statement of unbounded nondeterminacy, and thanks to this unbounded nondeterminacy, the program can generate <u>any</u> function f .

But this is not the solution Jayadev Misra was interested in. When we identify with the natural numbers a sequence of events, we can associate with the pair

(p,q) a message, put into the buffer at event no. p, and taken outof the buffer at event no. q. In our program these events are guarded by the mutually exclusive guards $n \notin S$ and $n \in S$ respectively, whereas for an unbounded buffer one would like the guards to be true and $S \neq \emptyset$ respectively —i.e. the only constraint being that you cannot take a message out of an empty buffer—.

We are now heading for a program that generates the (p,q) pairs in the order of increasing q. Again the elements in S correspond to the messages in the buffer, but instead of a q —the precise event number at which to leave the buffer— each element contains an index with the weaker connotation that the element with the smaller index leaves the buffer before the element with the larger index.

The new $P0$ is

$P0'$: $\langle \forall p,q :: M.p.q \wedge q < n \equiv$ "(p,q) has been generated"$\rangle$

and is initially established as before by

$n := 0$.

Besides the local variable S, which is now of type Set of (nat, nat) , there is a variable ind used to ensure that each element entering the buffer gets a larger index than ever left the buffer. The new $P1$ is

$P1'$ S is the smallest set satisfying

$$\langle \forall p :: p < n \wedge n \leq f.p \Rightarrow \langle \exists j :: ind \leq j \wedge (p,j) \in S \rangle \rangle$$

It is initially established by $S, ind := \emptyset, 0$.

The following program — Misra's, that is — does the job

```
|[ var n, ind : nat = 0,0
; var S: Set of (nat, nat)
; do S ≠ ∅ →
      |[ var r, i : nat
      ; solve r, i: (r, i) ∈ S ∧
                    ⟨∀p, j: (p, j) ∈ S : i ≤ j⟩
      ; generate (r, n)
      ; S := S - {(r, i)}; ind := i ; n := n+1
      ]|
  [] true →
      |[ var i: nat
      ; solve i: ind < i
```

```
        ; S := 𝒮 + {(n, i)} ; n := n+1
        ]|
   od
]|        .
```

With the guards not excluding each other, a new —and desired, see above— nondeterminacy has been introduced; progress requires that it be resolved fairly, and this requirement suffices. Fairness of the buffer is now shown by showing that with some element (p, j) in $\mathcal{S}$, only a finite number of selections of the first guarded command is possible before (p, j) itself is removed from $\mathcal{S}$. A slight complication is caused by the fact that several elements in $\mathcal{S}$ may have the same index, but it is overcome in the standard fashion by the lexical order on the pair

$$j - ind, \langle \underline{N} r : (r, ind) \in \mathcal{S} \rangle$$

and observing
(i) that it is bounded from below by $(0,0)$ as long as $(p, j) \in \mathcal{S}$
(ii) it is decreased by the first alternative
(iii) it is left unchanged by the second alternative

To show that also the second program can generate *any* f, we resolve its nondeterminacy by replacing in the second alternative

(i) the guard $true$ by $n < f.n$

(ii) the equation i: $ind < i$ for instance by (the stronger) i: $i = f.n$. (After the replacements, we can add the conjunct $ind \leq n$ to the invariant.)

Austin, 3 December 1995

prof. dr. Edsger W. Dijkstra
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78750-1188
USA