Eventvariables: an implementation with semaphores

0. Specification

An eventvariable is an integer variable x (say) on which only two operations are defined, which I shall call await.x and cause.x. The effect of these operations can be specified operationally —which is adequate for my purpose— as follows; here, h is an auxiliary variable of the process in which await.x occurs:

await.x: $\langle\, h := x \,\rangle$; $\langle\, [h<x] \,\rangle$

cause.x: $\langle\, x := x+1 \,\rangle$

legenda: A statement enclosed by $\langle \cdots \rangle$ must be implemented as an atomic action. For boolean expression B the statement $[B]$ ("wait until B") can be executed only if B is true, i.e. the process executing $[B]$ is blocked as long as B is false; otherwise, the net effect of $[B]$ is a skip.

□

It is not so surprising that await.x consists of 2 atomic statements: the effect of cause.x is different for processes that have initiated await.x and for processes that have not yet initiated await.x.

Every process has its own local auxiliary variable h; when necessary I shall use h_p for the auxiliary variable for process p. In what follows dummy p ranges over all processes.

The initial value of x is irrelevant, and so are the initial values of the h's. It is, therefore, safe to assume that, initially, $h_p < x$ for all p. By virtue of the above specification we now have that

Q: $\quad (\forall p :: h_p \leq x)$

is an invariant of the system. We even have, for all p:

$$h_p < x \;\vee\; \text{“process } p \text{ is blocked in await.}x\text{”}.$$

1. Implementation

All we have to do is to implement the guard $[h<x]$. Such a guard may be safely omitted if it has $h<x$ as a precondition. So, we are heading for a program of the following shape:

await.x: $\quad \langle h := x \rangle \;;$ ---?--- $\{h<x\}$

We use the (standard) technique of the split binary binary semaphore. It is not necessary to introduce a component semaphore for every individual condition $h<x$: first, we have $(\forall p :: h_p < x) \Rightarrow h_q < x$, for every q, and, second, we have that cause.x establishes $(\forall p :: h_p < x)$ because of the invariance of Q.

So, we introduce a semaphore s with an associated integer variable b, with the following invariant:

$$s = 0 \;\vee\; ((\forall p :: h_p < x) \wedge b > 0).$$

Initially we set $s=0 \wedge b=0$, which satisfies the invariant.

Moreover, according to the rules of the trade, we introduce a semaphore m (the "neutral" component), initially $m=1$, with accompanying invariant:

$$m=0 \quad \vee \quad (\exists p :: h_p = x) \quad \vee \quad b=0 \ .$$

m and s now constitute a split binary semaphore: $0 \leq m+s \leq 1$ will be invariant as well.

From the theory of semaphores I recall that, for every invariant of the shape $s=0 \vee \mathcal{B}$, we may <u>use</u> that $\mathcal{B}$ is a valid postassertion of every $P.s$ operation — i.e. we may write $P.s\,\{\mathcal{B}\}$ — provided that we <u>prove</u> that $\mathcal{B}$ is a valid preassertion of every $V.s$ operation — i.e. we must show that we may write $\{\mathcal{B}\}\,V.s$ —. Using this we now can construct programs for await.x and cause.x as follows, entirely according to the rules of the trade:

await.x:

```
    P.m
;   h := x
;   { h = x }
    b := b+1
;   { (∃p :: h_p = x) }
    V.m
;   P.s
    { (∀p :: h_p < x) ∧ b > 0 }
;   { hence, also: h < x }
    b := b-1
;   if b > 0 → V.s  □  b = 0 → V.m fi
```

cause.x P.m
; x := x + 1
; { (∀p :: h_p < x) }
if b > 0 → V.s [] b = 0 → V.m fi

In these programs both x and the variables h can now be considered as auxiliary variables and they can be eliminated. Doing so and omitting the annotation as well we obtain:

await.x: P.m ; b := b+1 ; V.m
; P.s ; b := b-1 ; if b > 0 → V.s [] b = 0 → V.m fi

cause.x: P.m ; if b > 0 → V.s [] b = 0 → V.m fi

Thus, every eventvariable can be implemented by means of two (binary) semaphores and one integer variable. So much for a nice application of the technique of the split binary semaphore.

apology: My writing hand is not in good shape today, nor is my handwriting.

□

Eindhoven, 17 november 1994
Rob R. Hoogerwoord