# A little program transformation

We consider two programs $p$ and $q$, sharing the variables $x.p$, $x.q$, and $h$. The initial state satisfies $\neg x.p \wedge \neg x.q \wedge (h \neq p \vee h \neq q)$. Program $p$ is a cyclic program with body

```
  $x.p, h := true, p$
; if $\neg x.q \rightarrow skip$
  [] $h \neq p \rightarrow skip$
  fi
; $CS.p$
; $x.p := false$ .
```

Each line of code is (considered to be) atomic. Program $q$ is Program $p$ with $p$ and $q$ interchanged.

Next, we deem to have shown that the above multiprogram is a correct mutual exclusion algorithm as far as its safety is concerned. We also deem to have shown that safety is preserved by "strengthening a guard". We will use the latter to transform the above mutual exclusion algorithm into another one in which the atomic multiple assignment no longer occurs.

To that end we introduce a fresh shared variable $y$ that will take over the rôle of $x$. Because safety is preserved by strengthening a guard, we require $y$ to satisfy $\neg y \Rightarrow \neg x$, or -- equivalently: $x \Rightarrow y$ --. This leaves us no choice on the relative order of the assignments $x := true$ and $y := true$. For program $p$ we obtain

```
  y.p := true
; x.p, h := true, p
; if ¬y.p  → skip
  ▯ h ≠ p  → skip
  fi
; CS.p
; y.p, x.p := false, false      ,
```

from which thought variable $x$ can now be removed

*

From experience we know that the requirement to use fine-grained atomic statements can considerably complicate the design and discussion of multiprograms. From the above example we can learn that perhaps sometimes the concern about the granularity is a completely separable one.

Austin, 17 October 1987

W.H.J. Feijen,
Department of Computer Sciences,
The University of Texas at Austin,
Austin, TX 78712 - 1188
U.S.A.