1. Algorithm 1 depicts the lock and unlock methods of the MCS queue algorithm. The loop in line 6 is the local spinning waiting for the lock to be released. The loop at line 10, however, seems redundant. We suggest ways for avoiding any loops in the code (except for the local spinning in line 6) by using an operation that accesses two base objects atomically. Recall that \texttt{compare\&swap}(o,exp,new) (CAS) writes the value \textit{new} to base object \textit{o} if its value is equal to \textit{exp}, and returns a success or failure indication; \texttt{double-compare\&swap} (DCAS) is similar to CAS, but operates on two independent base objects.

For each of the solutions we describe, either apply the solution and write a new pseudocode for the lock and unlock methods that avoids any loops (except for line 6), or explain exactly why a loop is unavoidable when using DCAS for this specific purpose.

(a) When executing lock, if tail points to pred (which is not null), use DCAS to set both the tail and pred to point to qnode.

(b) When executing unlock, if qnode.next is null, use DCAS to set both qnode.next and the tail to null.

\begin{algorithm}
\begin{tabular}{ll}
\textbf{Algorithm 1} MCS queue & \\
\textbf{struct} Qnode \{ & \\
  boolean locked = false, & \\
  Qnode next = null \} & \\
\textbf{function} LOCK(Qnode qnode) & \\
2: & \\
3: \textbf{if} pred\neq\text{null} \textbf{then} & \\
4: & \\
5: qnode.locked = true & \\
6: \textbf{while} qnode.locked \textbf{do wait} & \\
\textbf{function} UNLOCK(Qnode qnode) & \\
7: & \\
8: \textbf{if} qnode.next==\text{null} \textbf{then} & \\
9: & \\
10: \textbf{if} \text{CAS}(\text{tail},qnode,\text{null}) \textbf{then return} & \\
11: qnode.next.locked = false & \\
\end{tabular}
\end{algorithm}
Algorithm 2 Adopt-commit with multi-writer registers

shared variables: Proposal, initially ⊥, $R_0$, $R_1$, initially 0
local variable preference

1: ADOPT-COMMIT(v)
2: $R_v := 1$
3: if Proposal $\neq ⊥$ then preference := Proposal
4: else
5: preference := v
6: Proposal := preference
7: if $R_{1-v} \neq ⊥$ then return (adopt, preference)
8: else return (commit, preference)

2. Consider the adopt-commit algorithm with multi-writer registers depicted in Algorithm 2. Prove that it satisfies graded agreement, i.e., if a process decides (commit, $y_i$) then all processes decide either (adopt, $y_i$) or (commit, $y_i$).

3. A stack object supports two types of operations: a push operation inserts a value to the head of the object; a pop operation removes a value from the head, or if the object is empty, returns an empty indication. Show that the consensus problem for three processes cannot be solved in an asynchronous system using only stack objects, if two processes may fail by crashing.

Submission date: 3/5/2011, in pairs. Please submit to Eshcar’s mailbox on the fifth floor.