A Fast, Scalable Mutual Exclusion Algorithm*

Jae-Heon Yang
Department of Computer Science
University of Maryland at College Park
College Park, Maryland 20742-3255

James H. Anderson
Department of Computer Science
The University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27599-3175

October 1993

Abstract

This paper is concerned with synchronization under read/write atomicity in shared memory multiprocessors. We present a new algorithm for N-process mutual exclusion that requires only read and write operations and that has $O(\log_2 N)$ time complexity, where "time" is measured by counting remote memory references. The time complexity of this algorithm is better than that of all prior solutions to the mutual exclusion problem that are based upon atomic read and write instructions; in fact, the time complexity of most prior solutions is unbounded. Performance studies are presented that show that our mutual exclusion algorithm exhibits scalable performance under heavy contention. In fact, its performance rivals that of the fastest queue-based spin locks based on strong primitives such as compare-and-swap and fetch-and-add. We also present a modified version of our algorithm that generates only $O(1)$ memory references in the absence of contention.

Keywords: Fast mutual exclusion, local spinning, mutual exclusion, read/write atomicity, scalability, synchronization primitives, time complexity.

CR Categories: D.4.1, D.4.2, F.3.1

---

*Preliminary version was presented at the Twelfth Annual ACM Symposium on Principles of Distributed Computing, Ithaca, New York, August 1993. Work supported, in part, by NSF Contracts CCR-9109497 and CCR-9216421 and by the Center for Excellence in Space Data and Information Sciences (CESDIS).

1 Introduction

The mutual exclusion problem is a paradigm for resolving conflicting accesses to shared resources and has been studied for many years, dating back to the seminal paper of Dijkstra [6]. In this problem, each of a set of processes repeatedly executes a program fragment known as its "critical section". Before and after executing its critical section, a process executes two other program fragments, its "entry section" and "exit section", respectively. The entry and exit sections must be designed so that at most one process executes its critical section at any time, and so that each process in its entry section eventually executes its critical section. The former is known as the *mutual exclusion* property, and the latter as the *starvation-freedom* property. In some variants of the problem, starvation-freedom is replaced by the weaker requirement of *livelock-freedom*: if some process is in its entry section, then some process eventually executes its critical section.

Most early solutions to the mutual exclusion problem required only minimal hardware support, specifically atomic read and write instructions. Although of theoretical importance, most such algorithms were judged to be impractical from a performance standpoint, leading to the development of solutions requiring stronger hardware support such as read-modify-write operations. The poor performance of early read/write algorithms stems partially from two factors. First, such algorithms are not scalable, i.e., performance degrades dramatically as the number of contending processes increases. Second, even in the absence of contention, such algorithms require a process contending for its critical section to execute many operations.

The second of these two problems has subsequently been addressed, specifically by Lamport in [9], where a read/write algorithm is given that requires only a constant number of operations per critical section acquisition in the absence of contention. Following the title of Lamport's paper, such algorithms have come to be known simply as "fast mutual exclusion algorithms". This designation is somewhat of a misnomer, as such algorithms are not necessarily fast in the *presence* of contention. In fact, the problem of designing a scalable algorithm requiring only read/write atomicity has remained open. In this paper, we present such an algorithm.

Many mutual exclusion algorithms that exhibit poor scalability do so, in part, because they require processes to busy-wait on shared variables that are remotely-accessible, i.e., that require a traversal of the global interconnect between processors and shared memory when accessed. Recently, several queue-based mutual exclusion algorithms based on primitives that are stronger than reads and writes have been proposed in which this type of busy-waiting is avoided [3, 7, 10]. These algorithms are based on the idea of "local spinning". In particular, processes busy-wait only on *locally accessible* shared variables that do not require a traversal of the global interconnect when accessed. Performance studies presented in [3, 7, 10] have established the importance of local spinning in ensuring good performance under heavy contention.

Although the notion of a *locally* accessible *shared* variable may at first seem counterintuitive, there are two architectural paradigms that support it. In particular, on distributed shared memory machines, a shared variable can be made locally accessible by storing it in a local portion of shared memory, and on cache-coherent machines, programs can be structured so that when a process busy-waits on a shared variable, that variable migrates to a local cache line. Note that, because distributed shared memory machines require static allocation of shared variables to processors, an algorithm that locally spins on such a machine will also locally spin on cache-coherent machines. The reverse is not necessarily true. For this reason, we adopt the stricter distributed shared memory model when distinguishing between local and remote shared memory accesses in the time complexity computations given in this paper. Alternative definitions of local and remote memory accesses based on cache-coherence are briefly considered in Section 7.

In a recent paper [2], Anderson presented a mutual exclusion algorithm that uses only local spins and that requires only atomic read and write operations. In his algorithm, each of N processes executes $O(N)$ remote operations to enter its critical section whether there is contention or not. All other previously published mutual exclusion algorithms that are based on atomic reads and writes employ global busy-waiting and hence induce an unbounded number of remote operations under heavy contention. Most such algorithms also require $O(N)$ remote operations in the absence of contention. Some exceptions to the latter include the algorithm given by Kessels in [8] and the previously mentioned one given by Lamport in [9]. Kessels' algorithm generates $O(\log_2 N)$ remote operations in the absence of contention, while Lamport's generates $O(1)$. A variant of Lamport's algorithm has recently been presented by Styer in [13]. Although Styer claims that his algorithm is more scalable than Lamport's, in terms of time complexity, they are actually very similar: both generate unbounded remote operations under heavy contention and $O(1)$ operations in the absence of contention. Styer's claims of scalability are predicated upon complexity calculations that ignore operations performed within busy-waiting constructs. Because the processes in his algorithm busy-wait on remote variables, such complexity calculations do not give a true indication of scalability. Another recent algorithm of interest is one given by Michael and Scott in [11]. Although this algorithm generates $O(N)$ remote memory references in the presence of contention and $O(1)$ in the absence of contention, it requires both full- and half-word reads and writes to memory, which is a level of hardware support more powerful than ordinary read/write atomicity.

In this paper, we present a new mutual exclusion algorithm that requires only atomic reads and writes and in which all spins are local. Our algorithm induces $O(\log_2 N)$ remote operations under any amount of contention, and thus is an improvement over the algorithm given by Anderson in [2]. We also present a modified version of this algorithm that requires only $O(1)$ remote operations in the absence of contention. Unfortunately, in this modified algorithm, worst-case complexity rises to $O(N)$. However, we argue that this $O(N)$ behavior is rare, occurring only when transiting from a period of high contention to a period of low contention. Under high contention, this modified algorithm induces only $O(\log_2 N)$ remote operations. It is worth noting that our algorithm and its variation are starvation-free, whereas some of the aforementioned algorithms are not.

The results of this paper suggest two important open problems. First, note that the queue-based algorithms in [3, 7, 10] require $O(1)$ remote operations, whether there is contention or not (actually the algorithms in [3, 7] have $O(1)$ complexity only if coherent caches are provided). As our algorithm requires $O(\log_2 N)$ remote operations, it is natural to ask whether this gap is due to a fundamental weakness of atomic reads and writes, or whether there is a mutual exclusion algorithm based on such operations that has $O(1)$ complexity (which seems doubtful). A significant step towards answering this question was recently presented by us in [15]. This new result involves programs with limited "write-contention". The *write-contention* of a concurrent program is the number of processes that may be simultaneously enabled to write the same shared variable. It is shown in [15] that for any N-process mutual exclusion algorithm with write-contention K, there is an execution involving only one process in which that process executes at least $\Omega(\log_K N)$ remote memory references in its entry section. This lower bound holds assuming any of a variety of synchronization primitives, including read, write, test-and-set, load-and-store, compare-and-swap, and fetch-and-add. Because our mutual exclusion algorithm is contained within the class of algorithms based on such primitives that have constant write-contention, this result implies that our algorithm is asymptotically optimal for this class. Because the execution that establishes the lower bound involves only one process, it follows that fast mutual exclusion requires arbitrarily high write-contention. All fast mutual exclusion algorithms for N processes presented to date (including the modified version of the algorithm presented in this paper) have write-contention N.

Despite the significance of the lower bound proved in [15], the general question posed in the previous paragraph remains unanswered. If it turns out that the lower bound for mutual exclusion under read/write atomicity is $\Omega(\log_2 N)$, then a second question arises: namely, is it possible to develop a mutual exclusion algorithm under read/write atomicity that requires $O(1)$ remote operations in the absence of contention and $O(\log_2 N)$ remote operations in the presence of contention? As mentioned above, the modified version of our algorithm *almost* achieves such bounds.

The rest of the paper is organized as follows. In Section 2, we present our model of concurrent programs. The above-mentioned mutual exclusion algorithm is then presented in Section 3, and its correctness proof is given in Section 4. In Section 5, we consider the modified version of the algorithm discussed above. In Section 6, we present results from performance studies conducted on the BBN TC2000 and Sequent Symmetry multiprocessors. These studies indicate that our mutual exclusion algorithm exhibits scalable performance under heavy contention. In fact, the performance of our algorithm rivals (and sometimes beats) that of the queue-based algorithms mentioned above. We end the paper with concluding remarks in Section 7.

2 Definitions

In this section, we present our model of concurrent programs and define the relations used in reasoning about such programs. A *concurrent program* consists of a set of processes and a set of variables. A *process* is a sequential program consisting of labeled statements. Each *variable* of a concurrent program is either private or shared. A *private variable* is defined only within the scope of a single process, whereas a *shared variable* is defined globally and may be accessed by more than one process. Each process of a concurrent program has a special private variable called its *program counter*: the statement with label k in process p may be executed only when the value of the program counter of p equals k. For an example of the syntax we employ for programs, see Figure 1.

A program's semantics is defined by its set of "fair histories". The definition of a fair history, which is given below, formalizes the requirement that each statement of a program is subject to weak fairness. Before giving the definition of a fair history, we introduce a number of other concepts; all of these definitions apply to a given concurrent program.

A *state* is an assignment of values to the variables of the program. One or more states are designated as *initial states*. If state u can be reached from state t via the execution of statement s, then we say that s is *enabled* at state t and we write $t \xrightarrow{s} u$. If statement s is not enabled at state t, then we say that s is *disabled* at t. A *history* is a sequence $t_0 \xrightarrow{s_0} t_1 \xrightarrow{s_1} \cdots$, where t_0 is an initial state. A history may be either finite or infinite; in the former case, it is required that no statement be enabled at the last state of the history. A history is *fair* if it is finite or if it is infinite and each statement is either disabled at infinitely many states of the history or is infinitely often executed in the history. Note that this fairness requirement implies that each continuously enabled statement is eventually executed. Unless otherwise noted, we henceforth assume that all histories are fair.

With regard to complexity, we assume that each shared variable is *local* to at most one process and is *remote* to all other processes. This assumption is reflective of a distributed shared memory model. We refer to a statement execution as an *operation*. An operation is *remote* if it accesses remote variables, and is *local* otherwise.

Following [5], we define safety properties using *unless* assertions and progress properties using *leads-to* assertions. Consider two predicates P and Q over the variables of a program. The assertion P *unless* Q holds iff for any pair of consecutive states in any history of the program, if $P \wedge \neg Q$ holds in the first state, then $P \vee Q$ holds in the second. If predicate P is initially true and if P *unless* *false* holds, then predicate

P is said to be an *invariant*. We say that predicate P *leads-to* predicate Q, denoted $P \mapsto Q$, iff for each history $t_0 \xrightarrow{s_0} t_1 \xrightarrow{s_1} \cdots$ of the program, if P is true at some state t_i, then Q is true at some state t_j where $j \geq i$.

3 Mutual Exclusion Algorithm

In this section, we present our mutual exclusion algorithm. First, we state more precisely the conditions that must be satisfied by such an algorithm. In the mutual exclusion problem, there are N processes, each of which has the following structure.

while *true* **do**
    Noncritical Section;
    Entry Section;
    Critical Section;
    Exit Section
od

It is assumed that each process begins execution in its noncritical section. It is further assumed that each critical section execution terminates. By contrast, a process is allowed to halt in its noncritical section. No variable appearing in any entry or exit section may be referred to in any noncritical or critical section (except, of course, program counters). A program that solves this problem is required to satisfy the mutual exclusion and starvation-freedom properties, given earlier in Section 1. As shown in Section 4, mutual exclusion can be stated formally as an invariant, and starvation-freedom as a leads-to property. We also require that each process in its exit section eventually enters its noncritical section; this requirement is trivially satisfied by our solution (and most others), so we will not consider it further.

As in [8], we first solve the mutual exclusion problem for two processes, and then apply our two-process solution in a binary arbitration tree to get an N-process solution. The two-process algorithm is depicted in Figure 1. The two processes are denoted u and v, which are assumed to be distinct, positive integer values.

The algorithm employs five shared variables, $C[u]$, $C[v]$, T, $P[u]$, and $P[v]$. Variable $C[u]$ ranges over $\{0, u, v\}$ and is used by process u to inform process v of its intent to enter its critical section. Observe that $C[u] = u \neq 0$ holds while process u executes its statements 2 through 12, and $C[u] = 0$ holds otherwise. Variable $C[v]$ is used similarly. Variable T ranges over $\{u, v\}$ and is used as a tie-breaker in the event that both processes attempt to enter their critical sections at the same time. The algorithm ensures that the two processes enter their critical sections according to the order in which they update T. Variable $P[u]$ ranges over $\{0, 1, 2\}$ and is used by process u whenever it needs to busy-wait. Note that $P[u]$ is waited on only by process u, and thus can be stored in a memory location that is locally accessible to process u (in which case all spins are local). Variable $P[v]$ is used similarly by process v.

Loosely speaking, the algorithm works as follows. When process u wants to enter its critical section, it informs process v of its intention by establishing $C[u] = u$. Then, process u assigns its identifier u to the tie-breaker variable T, and initializes its spinning location $P[u]$. If process v has not shown interest in entering its critical section, in other words, if $C[v] = 0$ holds when u executes statement 4, then process u proceeds directly to its critical section. Otherwise, u reads the tie-breaker variable T. If $T \neq u$, which implies that $T = v$, then u can enter its critical section, as the algorithm prohibits v from entering its critical section when $C[u] = u \ \wedge \ T = v$ holds (recall that ties are broken in favor of the first process to update T). If $T = u$ holds, then either process v executed statement 2 before process u, or process v has executed statement 1 but not statement 2. In the first case, u should wait until v exits its critical section, whereas, in the second case, u should be able to proceed to its critical section. This ambiguity is resolved by having

```
shared var C : array[u, v] of {0, u, v};
           P : array[u, v] of 0..2;
           T : {u, v}
initially  C[u] = 0 ∧ C[v] = 0 ∧ P[u] = 0 ∧ P[v] = 0
```

```
process u
while true do
   0: Noncritical Section;
   1: C[u] := u;
   2: T := u;
   3: P[u] := 0;
   4: if C[v] ≠ 0 then
   5:    if T = u then
   6:       if P[v] = 0 then
   7:          P[v] := 1 fi;
   8:       while P[u] = 0 do /* null */ od;
   9:       if T = u then
  10:          while P[u] ≤ 1 do /* null */ od fi
         fi
      fi;
  11: Critical Section;
  12: C[u] := 0;
  13: if T ≠ u then
  14:    P[v] := 2 fi
od
```

```
process v
while true do
   0: Noncritical Section;
   1: C[v] := v;
   2: T := v;
   3: P[v] := 0;
   4: if C[u] ≠ 0 then
   5:    if T = v then
   6:       if P[u] = 0 then
   7:          P[u] := 1 fi;
   8:       while P[v] = 0 do /* null */ od;
   9:       if T = v then
  10:          while P[v] ≤ 1 do /* null */ od fi
         fi
      fi;
  11: Critical Section;
  12: C[v] := 0;
  13: if T ≠ v then
  14:    P[u] := 2 fi
od
```

Figure 1: Two-process mutual exclusion algorithm.

process u execute statements 6 through 10. Statements 6 and 7 are executed by process u to release process v in the event that it is waiting for u to update the tie-breaker variable (i.e., v is busy-waiting at statement 8). Statements 8 through 10 are executed by u to determine which process updated the tie-breaker variable first. Note that $P[u] \geq 1$ implies that v has already updated the tie-breaker, and $P[u] = 2$ implies that v has finished its critical section. To handle these two cases, process u first waits until $P[u] \geq 1$ (i.e., until v has updated the tie-breaker), re-examines T to see which process updated T last, and finally, if necessary, waits until $P[u] = 2$ (i.e., until process v finishes its critical section).

After executing its critical section, process u informs process v that it is finished by establishing $C[u] = 0$. If $T = v$, in which case process v is waiting to enter its critical section, then process u updates $P[v]$ in order to terminate v's busy-waiting loop.

As discussed above, the N-process case is solved by applying the above two-process algorithm in a binary arbitration tree. The resulting algorithm is shown in Figure 2. In this figure, process identifiers range over $\{0..N-1\}$, and for notational convenience, N is assumed to be a power of 2. Associated with each link in the tree is an entry section and an exit section. The entry and exit sections associated with the two links connecting a given node to its sons constitute a two-process mutual exclusion algorithm. Initially, all processes start at the leaves of the tree. To enter its critical section, a process is required to traverse a path

```
shared var  C : array[0..log₂ N − 1, 0..N − 1] of −1..N − 1;
            P : array[1..log₂ N − 1, 0..N − 1] of 0..2;
            T : array[0..log₂ N − 1, 0..N − 1] of 0..N − 1
initially   (∀j, i : 0 ≤ j < log₂ N :: C[j, i] = −1 ∧ P[j, i] = 0)
define      (∀l : 0 ≤ l < N :: comp(l) ≡ if even(l) then l + 1 else l − 1 fi)

process i
private var rival : 0..N;
            j : 0..log₂ N;
            k : 0..(N − 1)/2;
            l : 0..N − 1
initially   j = 0 ∧ k = i/2 ∧ l = i

while true do
   0: Noncritical Section;
   1: while j < log₂ N do
   2:    C[j, l] := i;
   3:    T[j, k] := i;
   4:    P[j, i] := 0;
   5:    rival := C[j, comp(l)];
   6:    if rival ≠ −1 then
   7:       if T[j, k] = i then
   8:          if P[j, rival] = 0 then
   9:             P[j, rival] := 1 fi;
  10:          while P[j, i] = 0 do /* null */ od;
  11:          if T[j, k] = i then
  12:             while P[j, i] ≤ 1 do /* null */ od fi
            fi
         fi;
  13:    j, k, l := j + 1, i/2^(j+2), i/2^(j+1)
      od;
  14: Critical Section;
  15: while j > 0 do
  16:    j, k, l := j − 1, i/2^j, i/2^(j−1)
  17:    C[j, l] := −1;
  18:    rival := T[j, k];
  19:    if rival ≠ i then
  20:       P[j, rival] := 2 fi
      od
od
```

Figure 2: N-process mutual exclusion algorithm.

from its leaf up to the root, executing the entry section of each link on this path. Upon exiting its critical section, a process traverses this path in reverse, this time executing the exit section of each link.

As seen in Figure 2, each variable of the two-process algorithm now has an additional subscript giving the appropriate level in the arbitration tree. As defined in Section 4, we use x/y to denote truncated integer division, i.e., $\lfloor x \div y \rfloor$. The expression $comp(l)$ is used to identify the C-variable of process i's two-process competitor at each level of the arbitration tree.

Although the concept of a binary arbitration tree is quite simple, one subtle problem does arise in the implementation. In particular, because each process waits on a local spin location at each level of the arbitration tree, it is important that each process knows the precise identity of any competitor within instances of the two process algorithm. For example, consider an algorithm with four processes, denoted 0 through 3. In the arbitration tree, process 0 first competes with process 1 at level 0, and then with one of processes 2 and 3 at level 1. At level 1 in the tree, both processes 2 and 3 access $C[1,1]$ and $T[1,0]$. However, at this level, each still has a unique spin location, namely $P[1,2]$ and $P[1,3]$, respectively. If process 0 encounters contention when competing at level 1, then it is important that process 0 knows precisely which of 2 or 3 it is competing against. Otherwise, for example, process 0 might update the spin location for process 2, $P[1,2]$, when in fact it is competing against process 3. This could result in a violation of either mutual exclusion or starvation-freedom. As seen in Figure 2, this "identity problem" is handled by means of the *rival* variable. Because of this problem, the two-process local-spin algorithm given by Anderson in [2] cannot be readily applied within an arbitration tree to get an N-process local-spin algorithm.

With regard to complexity, note that if variable $P[i]$ is local to process i in the two process algorithm, then process i executes a constant number of remote operations in its two-process entry and exit sections. It follows that, in the N-process algorithm, each process executes $O(\log_2 N)$ remote operations in its (N-process) entry and exit sections.

4 Correctness Proof

In this section, we prove that the mutual exclusion and starvation-freedom properties hold for N-process mutual exclusion algorithm of Figure 2. A correctness proof for the simpler two-process algorithm is presented in [14]. We begin by presenting definitions and notational conventions that will be used in the remainder of the paper. As in Section 3, we assume that the number of processes N is a power of 2.

Notational Conventions: Unless specified otherwise, we assume that i, p, and q range over $\{0..N-1\}$ and that $0 \leq level < log_2 N$ holds. We denote statement number k of process i as $k.i$. We use x/y to denote truncated integer division, i.e., $\lfloor x \div y \rfloor$. Let S be a subset of the statement labels in process i. Then, $i@\{S\}$ holds iff the program counter for process i equals some value in S. The following is a list of symbols we will use ordered by increasing binding power: $\equiv$, $\mapsto$, $\Rightarrow$, $\vee$, $\wedge$, $(=, \neq, >, <, \geq, \leq)$, $+$, $(\cdot, /)$, $\neg$, $(., @)$. The symbols enclosed in parentheses have the same priority. We sometimes use parentheses to override this binding rule. □

The function *comp* defined in Figure 2 has the following properties.

$$comp(comp(x)) = x \qquad \text{(C0)}$$

(C0) directly follows from the definition of *comp*. □

$$p/2^j = comp(q/2^j) \;\Rightarrow\; comp(p/2^j) = q/2^j \tag{C1}$$

The antecedent implies that $comp(p/2^j) = comp(comp(q/2^j))$ holds. By (C0), the consequent follows. □

$$p/2^{j+1} = q/2^{j+1} \;\Rightarrow\; p/2^j = q/2^j \;\vee\; p/2^j = comp(q/2^j) \tag{C2}$$

Let x denotes the value of $p/2^{j+1}$. By the definition of $p/2^{j+1}$, $x \cdot 2^{j+1} \leq p < x \cdot 2^{j+1} + 2^{j+1}$. (Recall that '/' denotes truncated integer division.) The antecedent implies that $x \cdot 2^{j+1} \leq p < x \cdot 2^{j+1} + 2^{j+1} \;\wedge\; x \cdot 2^{j+1} \leq q < x \cdot 2^{j+1} + 2^{j+1}$. Dividing by 2^j yields $2x \leq p/2^j < 2x + 2 \;\wedge\; 2x \leq q/2^j < 2x + 2$. This implies that $(p/2^j = 2x \;\vee\; p/2^j = 2x + 1) \;\wedge\; (q/2^j = 2x \;\vee\; q/2^j = 2x + 1)$ holds. The consequent follows from the definition of $comp(q/2^j)$. □

$$p/2^j = q/2^j \;\vee\; p/2^j = comp(q/2^j) \;\Rightarrow\; p/2^{j+1} = q/2^{j+1} \tag{C3}$$

(C3) directly follows from the definition of *comp*. □

To facilitate the presentation, we define the following predicate.

$$ME(level) \;\equiv\; (\forall i :: (\mathbf{N}p : p/2^{level} = i/2^{level} :: p.j \geq level) \;\leq\; 1)$$

The following invariants, which are stated without proof, follow directly from the program text.

invariant $\quad i@\{0\} \;\Rightarrow\; (i.j = 0)$ (I0)

invariant $\quad i@\{2..13, 17..20\} \;\Rightarrow\; (0 \leq i.j < \log_2 N)$ (I1)

invariant $\quad i@\{14\} \;\Rightarrow\; (i.j = \log_2 N)$ (I2)

invariant $\quad i.k = i/2^{i.j+1} \;\wedge\; i.l = i/2^{i.j}$ (I3)

We next prove several invariants on the relation between the program variables and *comp*.

invariant $\quad C[level, i/2^{level}] = p \;\wedge\; p \neq -1 \;\Rightarrow\; p/2^{level} = i/2^{level}$ (I7)

Initially $(\forall j, i :: C[j,i] = -1)$ holds, and hence (I7) is true. To prove that (I7) is not falsified, it suffices to consider only those statements that may establish the antecedent. The antecedent may be established only by statement $2.p$. Statement $2.p$ establishes the antecedent only if $p.j = level \;\wedge\; p.l = i/2^{level}$ holds. By (I3), this implies that $p.j = level \;\wedge\; p/2^{level} = i/2^{level}$ holds. Thus, statement $2.p$ preserves (I7). □

invariant $\quad C[level, comp(i/2^{level})] = p \;\wedge\; p \neq -1 \;\Rightarrow\; p/2^{level} = comp(i/2^{level})$ (I8)

The proof is similar to that given for (I7). □

invariant $\quad i.j = level \;\wedge\; i@\{6..9\} \;\Rightarrow\; i.rival = -1 \;\vee\; i.rival/2^{level} = comp(i/2^{level})$ (I9)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I9) is true. To prove that (I9) is not falsified, it suffices to consider only those statements that may establish the antecedent or falsify the consequent. The antecedent may be established only by statements $5.i$, $13.i$, and $16.i$. Statement $5.i$ establishes the antecedent when $i.j = level$ holds. By (I3), $i.j = level \ \wedge\ i.l = comp(i/2^{level})$ holds. (I8) implies that statement $5.i$ also establishes the consequent in that case. By the same reason, $5.i$ does not falsify the consequent when the antecedent holds. $\neg i@\{6..9\}$ holds after the execution of statements $13.i$ and $16.i$. Although statement $18.i$ may falsify the consequent, it establishes $i@\{19\}$. $\Box$

invariant $\quad (i.j > level \ \vee\ i.j = level \ \wedge\ i@\{4..20\}) \ \wedge\ T[level, i/2^{level+1}] = p \ \Rightarrow$
$p/2^{level+1} = i/2^{level+1}$ (I10)

Initially $(\forall i :: i.j = 0 \ \wedge\ i@\{0\})$ holds, and hence (I10) is true. The antecedent may be established only by statements $1.i$, $3.i$, $13.i$, $16.i$, and $3.p$. Statement $1.i$ establishes $i@\{14\}$ only if $i.j = \log_2 N$ holds, which implies that $i.j > level$. Statement $3.i$ establishes the antecedent only if $i.j = level$ holds. (I3) implies that statement $3.i$ establishes $T[level, i/2^{level+1}] = i$ in that case. Because $i/2^{level+1} = i/2^{level+1}$ holds, this implies that statement $3.i$ preserves (I10). Statement $13.i$ establishes $i.j > level$ only when $i.j = level \ \wedge\ i@\{13\}$ holds. Statement $16.i$ establishes $i.j = level$ only when $i.j > level$ holds. It follows that, although statements $1.i$, $3.i$, and $16.i$ may preserve the antecedent, they does not establish it.

Statement $3.p$ may establish the antecedent only if $p.j = level \ \wedge\ p.k = i/2^{level+1}$ holds. By (I3), this implies that $p/2^{level+1} = i/2^{level+1}$ holds. Thus, statement $3.p$ preserves (I10). $\Box$

invariant $\quad i.j = level \ \wedge\ i@\{19, 20\} \ \Rightarrow\ i.rival/2^{level+1} = i/2^{level+1}$ (I11)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I11) is true. The antecedent may be established only by statement $18.i$. Statement $18.i$ establishes the antecedent only if $i.j = level \ \wedge\ i@\{18\}$ holds. (I3) and (I10) imply that statement $18.i$ establishes the consequent in that case. $\Box$

Mutual Exclusion

We next prove that the following assertion is an invariant, which implies, by (I2), that the mutual exclusion property holds.

$$(\forall\ n : 0 \le n \le \log_2 N :: ME(n)) \tag{G0}$$

We use an induction on n in our proof. The induction base $ME(0)$ is an invariant by the definition of ME. Thus, it suffices to prove that the following assertion holds.

$$(\forall\ level : 0 \le level < \log_2 N :: ME(level) \ \Rightarrow\ ME(level+1)) \tag{G1}$$

Next, we prove several assertions that are needed to establish $ME(level+1)$. In these proofs, we assume that $ME(level)$ is an invariant.

invariant $\quad C[level, p/2^{level}] = p \ \Rightarrow\ p.j > level \ \vee\ p.j = level \ \wedge\ p@\{3..14, 17\}$ (I12)

Initially $(\forall\ level, i :: C[level, i] = -1)$ holds, and hence, because p ranges over $\{0..N-1\}$, (I12) is true. The antecedent is established only by statement $2.p$ when $p.j = level$ holds. Statement $2.p$ establishes

$p.j = level \ \wedge \ p@\{3\}$. Only statements 13.$p$, 14.$p$, 16.$p$, and 17.$p$ may falsify the consequent. When statement 13.p falsifies $p.j = level \ \wedge \ p@\{3..14, 17\}$, it establishes $p.j > level$. Statement 14.p does not falsify the consequent, because, by (I2), $p@\{14\} \ \Rightarrow \ p.j > level$ holds (recall that $level < \log_2 N$). When statement 16.p falsifies $p.j > level$, it establishes $p.j = level \ \wedge \ p@\{17\}$. Statement 17.$p$ may falsify the consequent only if $p.j = level$ holds. In this case, by (I3), statement 17.p also falsifies the antecedent. $\Box$

invariant $\quad p.j > level \ \vee \ p.j = level \ \wedge \ p@\{3..14, 17\} \Rightarrow \ C[level, p/2^{level}] = p \qquad$ (I13)

Initially $(\forall i :: i.j = 0 \ \wedge \ i@\{0\})$ holds, and hence (I13) is true. Only statements 1.p, 2.p, 13.p, and 16.p may establish the antecedent. The consequent may be falsified by statements 2.i or 17.i. Statement 1.p establishes $p@\{14\}$ only if $p.j = \log_2 N$ holds. This implies $p.j > level$. Thus, although statement 1.p may preserve the antecedent, it does not establish it. Statement 2.p may establish the antecedent only when $p.j = level$ holds. In this case, by (I3), it also establishes the consequent. Statement 13.p establishes $p.j > level$ only if $p.j = level \ \wedge \ p@\{13\}$ holds. Thus, although statement 13.p may preserve the antecedent, it does not establish it. Statement 16.p establishes $p.j = level \ \wedge \ p@\{17\}$ only if $p.j > level$ holds. Thus, it does not establish the antecedent. The consequent may be falsified by statements 2.i or 17.i only if $i.l = p/2^{level} \ \wedge \ i.j = level$ holds. This implies, by (I3), that $p/2^{level} = i/2^{level} \ \wedge \ i.j = level$ holds. If $i \neq p$ holds, then, by $ME(level)$, $p.j < level$, which implies that the antecedent of (I13) does not hold. If $i = p$, then statement 2.i does not falsify the consequent, and when statement 17.i falsifies the consequent, it also falsifies the antecedent. $\Box$

invariant $\quad C[level, p/2^{level}] = -1 \ \Rightarrow \ (\forall i : i/2^{level} = p/2^{level} :: i.j < level \ \vee$
$\quad i.j = level \ \wedge \ i@\{0, 1, 2, 15, 16, 18, 19, 20\}) \qquad$ (I14)

Assume, for the sake of contradiction, that $(C[level, p/2^{level}] = -1) \ \wedge \ i/2^{level} = p/2^{level} \ \wedge \ (i.j > level \vee i.j = level \wedge i@\{3..14, 17\})$ holds. Then, by (I13), that $(C[level, p/2^{level}] = -1) \wedge (C[level, i/2^{level}] = i) \ \wedge \ i/2^{level} = p/2^{level}$ holds. This is a contradiction. Thus, (I14) is an invariant. $\Box$

invariant $\quad (\forall i : i/2^{level} = p/2^{level} :: i.j < level \ \vee \ i.j = level \ \wedge \ i@\{0, 1, 2, 15, 16, 18, 19, 20\}) \ \Rightarrow$
$\quad C[level, p/2^{level}] = -1 \qquad$ (I15)

Assume, for the sake of contradiction, that $(\forall i : i/2^{level} = p/2^{level} :: i.j < level \ \vee \ i.j = level \ \wedge \ i@\{0, 1, 2, 15, 16, 18, 19, 20\}) \ \wedge \ C[level, p/2^{level}] = r \ \wedge \ r \neq -1$ holds. Then, by (I7), $r/2^{level} = p/2^{level}$ holds, and $r.j > level \ \vee \ (r.j = level \ \wedge \ r@\{3..14, 17\})$ holds by (I12). This is a contradiction. Thus, (I15) is an invariant. $\Box$

invariant $\quad (p.j > level \ \vee \ p.j = level \ \wedge \ p@\{4..20\}) \ \wedge \ q/2^{level} = comp(p/2^{level}) \ \wedge \ (q.j > level \ \vee$
$\quad q.j = level \ \wedge \ q@\{4..20\}) \ \Rightarrow \ T[level, p/2^{level+1}] = p \ \vee \ T[level, p/2^{level+1}] = q \qquad$ (I16)

Initially $(\forall i :: i.j = 0 \ \wedge \ i@\{0\})$ holds, and hence (I16) is true. The antecedent is established only by statements 1.p, 1.q, 3.p, 3.q, 13.p, 13.q, 16.p, and 16.q. As in the proof on (I13), although statement 1.p may preserve the antecedent, it does not establish it. Similar reasoning applies to 1.q. Statement 3.p establishes the antecedent only when $p.j = level$ holds. By (I3), this implies that the consequent is established. Statement 3.q establishes the antecedent only when $q.j = level$ holds. (I3) and $q/2^{level} = comp(p/2^{level})$ implies $q.k = p/2^{level+1}$. In this case, statement 3.q establishes the consequent. As in the proof of (I13), although statements 13.p and 16.p may preserve the antecedent, they do not establish it. Similar reasoning applies to $13.q$ and $16.q$. (I3) implies that the consequent may be falsified only by statement $3.i$, where $i/2^{level+1} = p/2^{level+1} \wedge i.j = level \wedge i \neq p \wedge i \neq q$ holds. Then, $i/2^{level} = p/2^{level} \vee i/2^{level} = comp(p/2^{level})$ holds by (C2). When $q/2^{level} = comp(p/2^{level})$ holds, $i.j = level \wedge i \neq p \wedge i \neq q \wedge (i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level})$ holds. This implies, by $ME(level)$, that $p.j < level \vee q.j < level$ holds, and hence, the antecedent does not hold. □

invariant $p.j = level \wedge p@\{6..9\} \wedge C[level, comp(p/2^{level})] = q \wedge q \neq -1 \Rightarrow$
$(p.rival = q \wedge q \neq -1) \vee T[level, p/2^{level+1}] = q \vee (q.j = level \wedge q@\{3\})$ (I17)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I17) is true. Only statements $5.p$, $13.p$, $16.p$, and $2.q$ may establish the antecedent. The consequent may be falsified only by statements $5.p$, $18.p$, $3.q$, and $3.i$, where $i \neq q$. Statement $5.p$ establishes the antecedent only if $p.j = level \wedge C[level, comp(p/2^{level})] = q \wedge q \neq -1$ holds. By (I3), this implies that statement $5.p$ also establishes $p.rival = q \wedge q \neq -1$. Statements $13.p$ and $16.p$ establish $\neg p@\{6..9\}$. Thus, they do not establish the antecedent. Statement $2.q$ may establish the antecedent only when $q.j = level$ holds. In this case, it also establishes $q.j = level \wedge q@\{3\}$. By (I3), statement $5.p$ may falsify the consequent only if $C[level, p/2^{level}] \neq q$ holds. Statement $18.p$ establishes $p@\{19\}$. Thus, statements $5.p$ and $18.p$ preserve (I17). Statement $3.q$ may falsify (I17) only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1 \wedge q.j = level \wedge q@\{3\}$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{3\}$ holds. By (C3), this implies that $q/2^{level+1} = p/2^{level+1} \wedge q.j = level \wedge q@\{3\}$ holds. In this case, (I3) implies that statement $3.q$ establishes $T[level, p/2^{level+1}] = q$. (I3) implies that statement $3.i$ may falsify $T[level, p/2^{level+1}] = q$ only if $i/2^{level+1} = p/2^{level+1} \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \vee i/2^{level} = comp(p/2^{level})) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. Note further that statement $3.i$ may falsify (I17) only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement $3.i$ may falsify (I17) only if $(i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level}) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. If $i/2^{level} = p/2^{level} \wedge i.j = level \wedge i@\{3\}$ holds, then, by $ME(level)$, this implies that $p.j = level$ does not hold. Otherwise, $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i \neq q$ holds. By $ME(level)$, this implies that $q.j < level$ holds. In that case, (I12) implies that $C[level, comp(p/2^{level})] = q$ does not hold. □

invariant $p.j = level \wedge (p@\{6\} \wedge p.rival \neq -1 \vee p@\{7..12\}) \wedge q/2^{level} = comp(p/2^{level}) \wedge$
$q.j = level \wedge q@\{19, 20\} \Rightarrow \quad q.rival = p \vee q.rival = q$ (I18)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I18) is true. The antecedent may be established only by statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $13.q$, $16.q$, and $18.q$. Only statements $5.q$ and $18.q$ may falsify the consequent. By (I3), statement $5.p$ may establish the antecedent only if $p.j = level \wedge C[level, comp(p/2^{level})] = i \wedge i \neq -1 \wedge q/2^{level} = comp(p/2^{level})$ holds for some i. By (I8), this implies that $C[level, comp(p/2^{level})] = i \wedge i/2^{level} = comp(p/2^{level}) \wedge i/2^{level} = q/2^{level}$ holds. By (I12), this implies that $(i.j > level \vee i.j = level \wedge i@\{3..14, 17\}) \wedge i/2^{level} = q/2^{level}$ holds. By $ME(level)$, this implies that $q.j = level$ does not hold. Statement $6.p$ may establish $p.j = level \wedge p@\{7..12\}$ only if $p.j = level \wedge p@\{6\} \wedge p.rival \neq -1$ holds. Thus, although statement $6.p$ may preserve, it does not establish the antecedent. Statements $13.p$, $16.p$, and $18.p$ establish $\neg p@\{6..12\}$, and hence, do not establish the antecedent. Statements $13.q$ and $16.q$ establish $\neg q@\{19, 20\}$, and hence, do not establish the antecedent. Statement $18.q$ establishes the antecedent only if $p.j = level \wedge (p@\{6\} \wedge p.rival \neq 0 \vee p@\{7..12\}) \wedge q.j = level \wedge q@\{18\}$ holds. By (I16), this implies that $T[level, p/2^{level+1}] = p \vee T[level, p/2^{level+1}] = q$ holds. In that case, by (I3), statement $18.q$ establishes the consequent. By the same reason, statement $18.q$ may falsify the consequent only if the antecedent does not hold. Finally, statement $5.q$ establishes $q@\{6\}$. $\Box$

invariant $p.j = level \ \wedge\ (p@\{6\}\ \wedge\ p.rival \neq -1\ \vee\ p@\{7..12\})\ \wedge\ q/2^{level} = comp(p/2^{level})\ \wedge$
$q.j = level\ \wedge\ q@\{20\}\ \Rightarrow\quad q.rival = p$ (I19)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I19) is true. The antecedent may be established only by statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $13.q$, $16.q$, and $19.q$. Only statements $5.q$ and $18.q$ may falsify the consequent. The reasoning for statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $13.q$, and $16.q$ is the same as that given in the proof of (I18). Statement $19.q$ establishes the antecedent only if $p.j = level\ \wedge\ (p@\{6\}\ \wedge\ p.rival \neq 0 \vee\ p@\{7..12\})\ \wedge\ q/2^{level} = comp(p/2^{level})\ \wedge\ q.j = level\ \wedge\ q@\{19\}\ \wedge\ q.rival \neq q$ holds. This implies, by (I18), that $q.rival = p$ holds. Finally, statements $5.q$ and $18.q$ establish $\neg q@\{20\}$. $\Box$

invariant $p.j = level\ \wedge\ (p@\{6\}\ \wedge\ p.rival \neq -1\ \vee\ p@\{7..12\})\ \wedge\ q/2^{level} = comp(p/2^{level})\ \wedge$
$q.j = level\ \wedge\ q@\{19\}\ \wedge\ q.rival = q\ \Rightarrow\quad T[level, p/2^{level+1}] = q$ (I20)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I20) is true. The antecedent may be established only by statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $5.q$, $13.q$, $16.q$, and $18.q$. Only statement $3.i$ may falsify the consequent. The reasoning for statements $5.p$, $6.p$, $13.p$, $16.p$, and $18.p$ is the same as that given in the proof of (I18). Statements $5.q$, $13.q$, and $16.q$ establish $\neg q@\{19\}$, and hence, do not establish the antecedent. If $q/2^{level} = comp(p/2^{level})$ holds, then, by (C3), $q/2^{level+1} = p/2^{level+1}$ also holds. Thus, (I3) implies that statement $18.q$ establishes the antecedent only if the consequent holds.

(I3) implies that statement $3.i$ may falsify $T[level, p/2^{level+1}] = q$ only if $i.j = level\ \wedge\ i/2^{level+1} = p/2^{level+1}\ \wedge\ i@\{3\}\ \wedge\ i \neq q$ holds. By (C2), $i.j = level\ \wedge\ (i/2^{level} = p/2^{level}\ \vee\ i/2^{level} = comp(p/2^{level}))\ \wedge\ i.j = level\ \wedge\ i@\{3\}\ \wedge\ i \neq q$ holds. Note further that statement $3.i$ may falsify (I20) only if $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement $3.i$ may falsify (I20) only if $(i/2^{level} = p/2^{level}\ \vee\ i/2^{level} = q/2^{level})\ \wedge\ i.j = level\ \wedge\ i@\{3\}\ \wedge\ i \neq q$ holds. If $i/2^{level} = p/2^{level}\ \wedge\ i.j = level\ \wedge\ i@\{3\}$ holds, then, by $ME(level)$, this implies that $p.j = level\ \wedge\ p@\{6..9\}$ does not hold. Otherwise, $i/2^{level} = q/2^{level}\ \wedge\ i.j = level\ \wedge\ i \neq q$ holds, which by $ME(level)$, implies that $q.j = level$ does not hold. $\Box$

invariant $(p.j > level\ \vee\ p.j = level\ \wedge\ p@\{5..18\})\ \wedge\ P[level, p] = 2\ \wedge\ C[level, comp(p/2^{level})] = q\ \wedge$
$q \neq -1\ \Rightarrow\ T[level, p/2^{level+1}] = q\ \vee\ q.j = level\ \wedge\ q@\{3\}$ (I21)

Initially $(\forall i :: i.j = 0\ \wedge\ i@\{0\})$ holds, and hence (I21) is true. The antecedent is established only by statements $1.p$, $4.p$, $13.p$, $16.p$, $20.i$, and $2.q$. The consequent may be falsified only by statements $3.q$, $13.q$, $16.q$, and $3.i$, where $i \neq q$. Statement $1.p$ establishes $p@\{14\}$ only if $p.j = \log_2 N$ holds. This implies $p.j > level$. Thus, although statement $1.p$ may preserve the antecedent, but it does not establish it. Statement $4.p$ may establish the antecedent only if $p.j = level$ holds. In this case, it also establishes $P[level, p] = 0$. Statement $13.p$ establishes $p.j > level$ only when $p.j = level\ \wedge\ p@\{13\}$ holds, and statement $16.p$ establishes $p.j = level$ only when $p.j > level$ holds. Thus, although statements $13.p$ and $16.p$ may preserve the antecedent, but they do not establish it.

(I11) implies that statement $20.i$ may establish $P[level, p] = 2$ only if $i.j = level\ \wedge\ i@\{20\}\ \wedge\ i/2^{level+1} = p/2^{level+1}$ holds. By (C2), this implies that $i.j = level\ \wedge\ i@\{20\}\ \wedge\ (i/2^{level} = p/2^{level}\ \vee\ i/2^{level} = comp(p/2^{level}))$ holds. Note further that statement $20.i$ may falsify (I21) only if $C[level, comp(p/2^{level})] = q\ \wedge\ q \neq -1$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement $20.i$ may falsify (I21) only if $(i/2^{level} = p/2^{level}\ \vee\ i/2^{level} = q/2^{level})\ \wedge\ i.j = level\ \wedge\ i@\{20\}$ holds. If

$i/2^{level} = p/2^{level} \wedge i.j = level \wedge i@\{20\}$ holds, then, by $ME(level)$, this implies that $p.j = level \wedge p@\{5..18\}$ does not hold. Otherwise, $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i@\{20\}$ holds, which, by $ME(level)$ and (I12), implies that $C[level, q/2^{level}] = q$ does not hold. Because $q/2^{level} = comp(p/2^{level})$, this implies that $C[level, comp(p/2^{level})] = q$ does not hold. Statement 2.q may establish the antecedent only if $q.j = level$ holds. In this case, it also establishes $q.j = level \wedge q@\{3\}$.

Statement 3.q may falsify (I21) only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1 \wedge q.j = level \wedge q@\{3\}$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{3\}$ holds. By (C3), this implies that $q/2^{level+1} = p/2^{level+1} \wedge q.j = level \wedge q@\{3\}$ holds. In that case, (I3) implies that statement 3.q establishes $T[level, p/2^{level+1}] = q$. Statements 13.$q$ and 16.q do not falsify $q.j = level \wedge q@\{3\}$. Finally, (I3) implies that statement 3.i may falsify $T[level, p/2^{level+1}] = q$ only if $i/2^{level+1} = p/2^{level+1} \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \vee i/2^{level} = comp(p/2^{level})) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. Note further that statement 3.i may falsify (I21) only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement 3.i may falsify (I21) only if $(i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level}) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. If $i/2^{level} = p/2^{level} \wedge i.j = level \wedge i@\{3\}$ holds, then this implies, by $ME(level)$, that $p.j = level \wedge p@\{6..9\}$ does not hold. Otherwise, $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i \neq q$ holds, which, by $ME(level)$ and (I12), implies that $C[level, q/2^{level}] = q$ does not hold. Because $q/2^{level} = comp(p/2^{level})$, this implies that $C[level, comp(p/2^{level})] = q$ does not hold. Thus, statement 3.i preserves (I21). □

invariant $\quad (p.j > level \vee p.j = level \wedge p@\{13..20\}) \wedge C[level, comp(p/2^{level})] = q \wedge q \neq -1 \Rightarrow$
$T[level, p/2^{level+1}] = q \vee q.j = level \wedge q@\{3\}$ (I22)

Initially $(\forall i :: i.j = 0 \wedge i@\{0\})$ holds, and hence (I22) is true. The antecedent may be established only by statements 1.p, 6.p, 7.p, 11.p, 12.p, 13.p, 16.p, and 2.q. Only statements 3.q, 13.q, 16.q, and 3.i, where $i \neq q$, may falsify the consequent. The reasoning for statements 1.p, 13.p, 16.p, 3.q, 13.q, and 16.q is the same as that given in the proof of (I21).

Statement 6.p may establish the antecedent only if $p.j = level \wedge p@\{6\} \wedge p.rival = -1$ holds. By (I17), this implies that $\neg(C[level, comp(p/2^{level})] = q \wedge q \neq -1) \vee T[level, p/2^{level+1}] = q \vee (q.j = level \wedge q@\{3\})$ holds. This implies that either the antecedent does not hold or the consequent holds after the execution of statement 6.p.

(I3) implies that statements 7.p and 11.p establish the antecedent only if $p.j = level \wedge T[level, p/2^{level+1}] \neq p \wedge p@\{7, 11\} \wedge C[level, comp(p/2^{level})] = q \wedge q \neq -1$ holds. By (I8), this implies that $p.j = level \wedge T[level, p/2^{level+1}] \neq p \wedge p@\{7, 11\} \wedge q/2^{level} = comp(p/2^{level}) \wedge C[level, q/2^{level}] = q$ holds. In that case, (I12) implies that $p.j = level \wedge T[level, p/2^{level+1}] \neq p \wedge p@\{7, 11\} \wedge q/2^{level} = comp(p/2^{level}) \wedge (q.j > level \vee q.j = level \wedge q@\{3..14, 17\})$ holds. By predicate calculus, this implies that $T[level, p/2^{level+1}] \neq p \wedge p.j = level \wedge p@\{7, 11\} \wedge q/2^{level} = comp(p/2^{level}) \wedge ((q.j > level \vee q.j = level \wedge q@\{4..17\}) \vee (q.j = level \wedge q@\{3\}))$ holds. This implies, by (I16), that the consequent holds.

Statement 12.p establishes the antecedent only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1 \wedge p.j = level \wedge p@\{12\} \wedge P[level, p] = 2$ holds. By (I21), this implies that the consequent holds. Statement 2.q may establish the antecedent only if $q.j = level$ holds. In this case, it also establishes $q.j = level \wedge q@\{3\}$.

(I3) implies that statement 3.i may falsify $T[level, p/2^{level+1}] = q$ only if $i/2^{level+1} = p/2^{level+1} \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \vee i/2^{level} = comp(p/2^{level})) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$ holds. Note further that statement 3.i may falsify (I22) only if $C[level, comp(p/2^{level})] = q \wedge q \neq -1$ holds. By (I8), this implies that $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement 3.i may falsify (I22) only if $(i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level}) \wedge i.j = level \wedge i@\{3\} \wedge i \neq q$

holds. If $i/2^{level} = p/2^{level} \wedge i.j = level \wedge i@\{3\}$ holds, then, this implies, by $ME(level)$, that $p.j = level \wedge p@\{13..20\}$ does not hold. Otherwise, $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i \neq q$ holds, which, by $ME(level)$ and (I12), implies that $C[level, q/2^{level}] = q$ does not hold. Because $q/2^{level} = comp(p/2^{level})$, this implies that $C[level, comp(p/2^{level})] = q$ does not hold. Thus, statement $3.i$ preserves (I22). □

invariant $\quad (\forall i :: (\mathbf{N}p : p/2^{level+1} = i/2^{level+1} :: p.j \geq level + 1) \leq 1)$ (I23)

Assume, for the sake of contradiction, that $p.j > level \wedge q.j > level \wedge p/2^{level+1} = i/2^{level+1} \wedge q/2^{level+1} = i/2^{level+1} \wedge p \neq -1 \wedge q \neq -1$ holds for some i. By $ME(level)$ and (C2), we can assume, that $p.j > level \wedge q.j > level \wedge p/2^{level} = i/2^{level} \wedge q/2^{level} = comp(i/2^{level}) \wedge p \neq -1 \wedge q \neq -1$ holds. This implies, by (I13), that $p.j > level \wedge q.j > level \wedge C[level, p/2^{level}] = p \wedge C[level, q/2^{level}] = q \wedge p/2^{level} = i/2^{level} \wedge q/2^{level} = comp(i/2^{level}) \wedge p \neq -1 \wedge q \neq -1$ holds. Then, by (C1), $p.j > level \wedge q.j > level \wedge C[level, comp(p/2^{level})] = q \wedge C[level, comp(q/2^{level})] = p \wedge p \neq -1 \wedge q \neq -1$ holds. By (I22), this implies that $T[level, p/2^{level+1}] = q \wedge T[level, q/2^{level+1}] = p$ holds, which is a contradiction. Thus, (I23) is an invariant. □

Observe that we have completed the proof of (G1), and thus proved that (G0) holds. Because $ME(level)$ has been proved to hold for any $level$, the invariants from (I12) to (I23) hold for any $level$.

Progress

We next prove a number of invariants that are needed to establish starvation-freedom.

invariant $\quad p.j = level \wedge (p@\{6\} \wedge p.rival \neq -1 \vee p@\{7..12\}) \wedge$
$(\forall i : i/2^{level} = comp(p/2^{level}) :: i.j < level \vee i.j = level \wedge i@\{0,1,2,15,16\}) \Rightarrow P[level, p] = 2$ (I24)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I24) is true. The antecedent may be established only by statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $13.i$, $14.i$, $15.i$, $16.i$, $19.i$, and $20.i$. Only statements $4.p$ and $9.i$ may falsify the consequent.

As in the proof of (I18), statement $5.p$ may establish the antecedent only if $i/2^{level} = comp(p/2^{level}) \wedge (i.j > level \vee i.j = level \wedge i@\{3..14, 17\})$ holds, which implies that the antecedent of (I24) does not hold. The reasoning for statements $6.p$, $13.p$, $16.p$, and $18.p$ is the same as that given in the proof of (I18).

(I2) implies that when statement $14.i$ establishes $i@\{15\}$, $i.j > level$ holds. Thus, statement $14.i$ does not establish the antecedent.

Statement $13.i$ establishes $i.j = level \wedge i@\{1\}$ only if $i.j < level$ holds. Statement $15.i$ may establish $i.j = level \wedge i@\{0\}$ only if $i.j = level \wedge i@\{15\}$ holds. Statement $16.i$ establishes $i.j < level$ only if $i.j = level \wedge i@\{16\}$ holds. Thus, although statements $13.i$, $15.i$, and $16.i$ may preserve the antecedent, they do not establish it.

Statement $19.i$ may establish the antecedent only if $p.j = level \wedge (p@\{6\} \wedge p.rival \neq -1 \vee p@\{7..12\}) \wedge i/2^{level} = comp(p/2^{level}) \wedge i.j = level \wedge i@\{19\} \wedge i.rival = i$ holds. This implies, by (I20), that $p.j = level \wedge (p@\{6\} \wedge p.rival \neq -1 \vee p@\{7..12\}) \wedge i/2^{level} = comp(p/2^{level}) \wedge i.j = level \wedge i@\{19\} \wedge T[level, p/2^{level+1}] = i$ holds. By (I13), this implies that $C[level, p/2^{level}] = p \wedge T[level, p/2^{level+1}] \neq p \wedge i/2^{level} = comp(p/2^{level}) \wedge i.j = level \wedge i@\{19\}$ holds. This implies, by (C1) and (C3), that $C[level, comp(i/2^{level})] = p \wedge T[level, i/2^{level+1}] \neq p \wedge i.j = level \wedge i@\{19\}$ holds. By (I22), this implies that $p.j = level \wedge p@\{3\}$ holds, which implies that the antecedent does not hold. Thus, statement $19.i$ does not establish the antecedent.

Statement $20.i$ establishes the antecedent only if $p.j = level \land (p@\{6\} \land i.rival \neq -1 \lor p@\{7..12\}) \land i/2^{level} = comp(p/2^{level}) \land i.j = level \land i@\{20\}$ holds. By (I19), this implies that $i.j = level \land i@\{20\} \land i.rival = p$ holds. In this case, statement $20.i$ establishes the consequent.

Statement $4.p$ establishes $p@\{5\}$. (I9) implies that statement $9.i$ may falsify the consequent only if $i.j = level \land i@\{9\} \land p/2^{level} = comp(i/2^{level})$ holds. By (C1), this implies that $i.j = level \land i@\{9\} \land i/2^{level} = comp(p/2^{level})$ holds, which implies that the antecedent does not hold. □

invariant $p.j = level \land p@\{8..10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{6..10\} \land T[level, q/2^{level+1}] = q \Rightarrow q.rival = p$ (I25)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I25) is true. The antecedent may be established only by statements $7.p$, $13.p$, $16.p$, $3.q$, $5.q$, $13.q$, and $16.q$. Only statements $5.q$ and $18.q$ may falsify the consequent. (I3) implies that statement $7.p$ may establish the antecedent only if $q/2^{level} = comp(p/2^{level}) \land T[level, p/2^{level+1}] = p$ holds. By (C3), this implies that $T[level, q/2^{level+1}] = p$ holds, which implies that the antecedent does not hold. $\neg p@\{8..10\}$ holds after the execution of statements $13.p$ and $16.p$. $\neg q@\{6..10\}$ holds after the execution of statements $3.q$, $13.q$, and $16.q$. Statement $5.q$ may falsify (I25) only if $p.j = level \land p@\{8..10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{5\}$ holds. By (I13), this implies that $C[level, p/2^{level}] = p \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{5\}$ holds. This implies, by (C1), that $C[level, comp(q/2^{level})] = p \land q.j = level \land q@\{5\}$ holds. In this case, (I3) implies that statement $5.q$ establishes the consequent. Finally, statement $18.q$ establishes $q@\{19\}$. □

invariant $p.j = level \land p@\{8..10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{10..12\} \land T[level, q/2^{level+1}] = q \Rightarrow P[level, p] \geq 1$ (I26)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I26) is true. The antecedent may be established only by statements $7.p$, $13.p$, $16.p$, $3.q$, $8.q$, $9.q$, $13.q$, and $16.q$. Only statement $4.p$ may falsify the consequent. The reasoning for statements $7.p$, $13.p$, $16.p$, $3.q$, $13.q$, and $16.q$ is similar to that given in the proof of (I25) Statement $8.q$ may establish the antecedent only if $p.j = level \land p@\{8..10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{8\} \land T[level, q/2^{level+1}] = q \land P[level, q.rival] \geq 1$ holds. By (I25), this implies that the consequent holds. Statement $9.q$ may establish the antecedent only if $p.j = level \land p@\{8..10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{9\} \land T[level, q/2^{level+1}] = q$ holds. By (I25), this implies that statement $9.q$ establishes the consequent. Finally, statement $4.p$ establishes $p@\{5\}$. □

invariant $p.j = level \land p@\{10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{10\} \Rightarrow P[level, p] \geq 1 \lor P[level, p] \geq 1$ (I27)

The antecedent implies, by (I16), that $p.j = level \land p@\{10\} \land q/2^{level} = comp(p/2^{level}) \land q.j = level \land q@\{10\} \land (T[level, p/2^{level+1}] = p \lor T[level, p/2^{level+1}] = q)$ holds. By (C1) and (C3), this implies that $p.j = level \land p@\{10\} \land q/2^{level} = comp(p/2^{level}) \land p/2^{level} = comp(q/2^{level}) \land q.j = level \land q@\{10\} \land (T[level, p/2^{level+1}] = p \lor T[level, q/2^{level+1}] = q)$ holds. By (I26), this implies that the consequent holds. □

invariant $i.j = level \land i@\{11, 12\} \Rightarrow P[level, i] \geq 1$ (I28)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I28) is true. The antecedent may be established only by statements

10.i, 13.i, and 16.i. Only statement 4.i may falsify the consequent. Statement 10.i establishes the antecedent only if the consequent holds. $\neg i@\{11,12\}$ holds after the execution of statements 4.i, 13.i, and 16.i. $\Box$

invariant $p.j = level \ \wedge\ p@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ q.j = level \ \wedge\ q@\{5..12\} \ \wedge$
$T[level, q/2^{level+1}] = q \ \Rightarrow\ P[level, q] = 0$ (I29)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I29) is true. The antecedent may be established only by statements 11.p, 13.p, 16.p, 3.q, 4.q, 13.q, and 16.q. Only statements 9.i and 20.i may falsify the consequent. (I3) implies that statement 11.p may establish the antecedent only if $p.j = level \ \wedge\ p@\{11\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ T[level, p/2^{level+1}] = p$ holds. By (C3), this implies that $T[level, q/2^{level+1}] = p$ holds, which implies that the antecedent does not hold. $\neg p@\{12\}$ holds after the execution of statements 13.p and 16.p. $\neg q@\{5..12\}$ holds after the execution of statements 3.q, 13.q, and 16.q. When statement 4.q establishes $q.j = level \ \wedge\ q@\{5\}$, it also establishes the consequent.

(I9) implies that statement 9.i may falsify the consequent only if $q/2^{level} = comp(p/2^{level}) \ \wedge\ q/2^{level} = comp(i/2^{level}) \wedge i.j = level \wedge i@\{9\}$ holds. By (C1), this implies that $p/2^{level} = i/2^{level} \wedge i.j = level \wedge i@\{9\}$ holds. In that case, $ME(level)$ implies that $p.j = level \wedge p@\{12\}$ does not hold. (I11) implies that statement 20.i may falsify the consequent only if $q/2^{level} = comp(p/2^{level}) \ \wedge\ q/2^{level+1} = i/2^{level+1} \ \wedge\ i.j = level \ \wedge\ i@\{20\}$ holds. By (C1), this implies that $comp(q/2^{level}) = p/2^{level} \wedge q/2^{level+1} = i/2^{level+1} \ \wedge\ i.j = level \ \wedge\ i@\{20\}$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \ \vee\ i/2^{level} = q/2^{level}) \ \wedge\ i.j = level \ \wedge\ i@\{20\}$ holds. If $i = p$, then statement 20.i establishes $\neg p@\{12\}$. If $i/2^{level} = p/2^{level} \wedge i \neq p \wedge i.j = level \wedge i@\{20\}$ holds, then $ME(level)$ implies that $p.j = level$ does not hold. If $i = q$, then statement 20.i establishes $\neg q@\{5..12\}$. If $i/2^{level} = q/2^{level} \ \wedge\ i \neq q \ \wedge\ i.j = level \ \wedge\ i@\{20\}$ holds, then $ME(level)$ implies that $q.j = level$ does not hold. $\Box$

invariant $\neg(p.j = level \ \wedge\ p@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ q.j = level \ \wedge\ q@\{12\})$ (I30)

Assume, for the sake of contradiction, that $p.j = level \ \wedge\ p@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ q.j = level \ \wedge\ q@\{12\}$ holds. By (I28), this implies that $p.j = level \ \wedge\ p@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ q.j = level \ \wedge\ q@\{12\} \ \wedge\ P[level, p] \geq 1 \ \wedge\ P[level, q] \geq 1$ holds. By (I16), this implies that $p.j = level \ \wedge\ p@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ q.j = level \ \wedge\ q@\{12\} \ \wedge\ P[level, p] \geq 1 \ \wedge\ P[level, q] \geq 1 \wedge\ (T[level, p/2^{level+1}] = p \ \vee\ T[level, p/2^{level+1}] = q)$ holds. In that case, (C1) and (C3) imply that $P[level, p] \geq 1 \ \wedge\ P[level, q] \geq 1 \ \wedge\ p.j = level \ \wedge\ p@\{12\} \ \wedge\ q.j = level \ \wedge\ q@\{12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ p/2^{level} = comp(q/2^{level}) \ \wedge\ (T[level, q/2^{level+1}] = q \ \vee\ T[level, p/2^{level+1}] = p)$ holds. By (I29), this implies that $P[level, p] \geq 1 \ \wedge\ P[level, q] \geq 1 \ \wedge\ (P[level, p] = 0 \ \vee\ P[level, q] = 0)$ holds, which is a contradiction. Thus (I30) is an invariant. $\Box$

invariant $p.j = level \ \wedge\ p@\{4..12\} \ \wedge\ q/2^{level} = comp(p/2^{level}) \ \wedge\ (q.j > level \ \vee\ q.j = level \ \wedge$
$q@\{13..20\}) \ \Rightarrow\ T[level, q/2^{level+1}] = p$ (I31)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I31) is true. The antecedent may be established only by statements 3.p, 13.p, 16.p, 1.q, 6.q, 7.q, 11.q, 12, q, 13.q, and 16.q. Only statement 3.i may falsify the consequent. Statement 3.p may establish the antecedent only if $p.j = level \ \wedge\ p@\{3\} \ \wedge\ q/2^{level} = comp(p/2^{level})$ holds. By (C3), this implies that $p.j = level \ \wedge\ p@\{3\} \ \wedge\ p/2^{level+1} = q/2^{level+1}$ holds. In this case, by (I3), statement 3.p establishes the consequent. Statements 13.p and 16.p preserve $\neg p@\{4..12\}$, and hence do not establish the antecedent. When statement 1.q establishes $q@\{14\}$, $q.j \geq \log_2 N$ holds, which implies that

$q.j > level$ holds. Thus, although statement $1.q$ may preserve the antecedent, it does not establish it.

Statement $6.q$ may establish the antecedent only if $p.j = level \wedge p@\{4..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{6\} \wedge q.rival = -1$ holds. By (I13), this implies that $p.j = level \wedge p@\{4..12\} \wedge C[level, p/2^{level}] = p \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{6\} \wedge q.rival = -1$ holds. By (C1), this implies that $p.j = level \wedge p@\{4..12\} \wedge C[level, comp(q/2^{level})] = p \wedge p \neq -1 \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{6\} \wedge q.rival = -1$ holds. In that case, (I17) implies that the consequent holds.

By (I3), statements $7.q$ and $11.q$ may establish the antecedent only if $p.j = level \wedge p@\{4..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{7, 11\} \wedge T[level, q/2^{level+1}] \neq q$ holds. By (C3), this implies that $p.j = level \wedge p@\{4..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{7, 11\} \wedge T[level, q/2^{level+1}] \neq q \wedge p/2^{level+1} = q/2^{level+1}$ holds. By (I16), this implies that $T[level, q/2^{level+1}] = p$ holds.

Statement $12.q$ may establish the antecedent only if $p.j = level \wedge p@\{4..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{12\} \wedge P[level, q] = 2$ holds. By (I13), this implies that $p.j = level \wedge p@\{4..12\} \wedge C[level, p/2^{level}] = p \wedge p \neq -1 \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{12\} \wedge P[level, q] = 2$ holds. By (C1), this implies that $p.j = level \wedge p@\{4..12\} \wedge C[level, comp(q/2^{level})] = p \wedge p \neq -1 \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{12\} \wedge P[level, q] = 2$ holds. In that case, by (I21), $T[level, q/2^{level+1}] = p$ holds. This implies that the consequent of (I31) holds. Statement $13.q$ establishes $q.j > level$ only when $q.j = level \wedge q@\{13\}$ holds, and statement $16.q$ establishes $q.j = level$ only when $q.j > level$ holds. Thus, although statements $13.q$ and $16.q$ may preserve the antecedent, but they do not establish it.

(I3) implies that statement $3.i$ may falsify $T[level, q/2^{level+1}] = p$ only if $i/2^{level+1} = p/2^{level+1} \wedge i.j = level \wedge i@\{3\} \wedge i \neq p$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \vee i/2^{level} = comp(p/2^{level})) \wedge i.j = level \wedge i@\{3\} \wedge i \neq p$ holds. Note further that statement $3.i$ may falsify (I31) only if $q/2^{level} = comp(p/2^{level})$ holds. Thus, statement $3.i$ may falsify (I31) only if $(i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level}) \wedge i.j = level \wedge i@\{3\} \wedge i \neq p$ holds. If $i/2^{level} = p/2^{level} \wedge i.j = level \wedge i \neq p$ holds, then $ME(level)$ implies that $p.j = level$ does not hold. If $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i@\{3\}$ holds, then, by $ME(level)$, this implies that $(q.j > level \vee q.j = level \wedge q@\{13..20\})$ does not hold. □

invariant $p.j = level \wedge p@\{5..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{19, 20\} \Rightarrow q.rival = p \vee P[level, p] = 0 \vee P[level, p] = 2$ (I32)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I32) is true. The antecedent is established only by statements $4.p$, $13.p$, $16.p$, $13.q$, $16.q$, and $18.q$. Only statements $5.q$, $18.q$, and $9.i$ may falsify the consequent. Statement $4.p$ may establish the antecedent only if $p.j = level$ holds. In this case, it also establishes $P[level, p] = 0$. Statements $13.p$ and $16.p$ preserve $\neg p@\{5..12\}$, and hence do not establish the antecedent. Statements $13.q$ and $16.q$ preserve $\neg q@\{19, 20\}$, and hence do not establish the antecedent. Statement $18.q$ may falsify (I32) only if $p.j = level \wedge p@\{5..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{18\}$ holds. By (I31), this implies that $q.j = level \wedge q@\{18\} \wedge T[level, q/2^{level+1}] = p$ holds. In this case, (I3) implies that statement $18.q$ establishes $q.rival = p$. Thus, statement $18.q$ preserves (I32). Although statement $5.q$ may falsify the consequent, it establishes $q@\{6\}$. (I9) implies that statement $9.i$ may falsify the consequent only if $q/2^{level} = comp(p/2^{level}) \wedge p/2^{level} = comp(i/2^{level}) \wedge i.j = level \wedge i@\{9\}$ holds. By (C1), this implies that $q/2^{level} = i/2^{level} \wedge i.j = level \wedge i@\{9\}$ holds. In that case, $ME(level)$ implies that $q.j = level \wedge q@\{19, 20\}$ does not hold. □

invariant $p.j = level \wedge p@\{5..12\} \wedge (\forall i : i/2^{level} = comp(p/2^{level}) :: i.j < level \vee i.j = level \wedge i@\{0..3, 15, 16\}) \Rightarrow P[level, p] = 0 \vee P[level, p] = 2$ (I33)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I33) is true. The antecedent may be established only by statements $4.p$, $13.p$, $16.p$, $13.i$, $14.i$, $15.i$, $16.i$, $19.i$, and $20.i$. Only statement $9.i$ may falsify the consequent. The reasoning for statements $4.p$, $13.p$, and $16.p$ is the same as in the proof of (I32). (I2) implies that when statement $14.i$ establishes $i@\{15\}$, $i.j > level$ holds. Thus, statement $14.i$ does not establish the antecedent.

Statement $13.i$ establishes $i.j = level \;\wedge\; i@\{1\}$ only if $i.j < level$ holds. Statement $15.i$ may establish $i.j = level \;\wedge\; i@\{0\}$ only if $i.j = level \;\wedge\; i@\{15\}$ holds. Statement $16.i$ establishes $i.j < level$ only if $i.j = level \;\wedge\; i@\{16\}$ holds. Thus, although statements $13.i$, $15.i$, and $16.i$ may preserve the antecedent, but they do not establish it.

Statements $19.i$ and $20.i$ establish the antecedent only if $p.j = level \;\wedge\; p@\{5..12\} \;\wedge\; i/2^{level} = comp(p/2^{level}) \;\wedge\; i.j = level \;\wedge\; i@\{19,20\}$ holds. By (I32), this implies that $i.j = level \;\wedge\; i@\{19,20\} \;\wedge\; (i.rival = p \;\vee\; P[level,p] = 0 \;\vee\; P[level,p] = 2)$ holds. If $i.rival \neq p$, then the consequent of (I33) holds, and is not falsified by statements $19.i$ and $20.i$. If $i.rival = p$, then statement $19.i$ establishes $i@\{20\}$, and statement $20.i$ establishes $P[level,p] = 2$.

(I9) implies that statement $9.i$ may falsify the consequent only if $p/2^{level} = comp(i/2^{level}) \;\wedge\; i.j = level \;\wedge\; i@\{9\}$ holds. This implies, by (C1), that $i/2^{level} = comp(p/2^{level}) \;\wedge\; i.j = level \;\wedge\; i@\{9\}$ holds. In that case, the antecedent does not hold. □

invariant $p.j = level \;\wedge\; (p@\{6\} \;\wedge\; p.rival \neq -1 \;\vee\; p@\{7..12\}) \;\wedge\; q/2^{level} = comp(p/2^{level}) \;\wedge$
$q.j = level \;\wedge\; q@\{19,20\} \;\Rightarrow\; q.rival = p$ (I34)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I34) is true. The antecedent may be established only by statements $5.p$, $6.p$, $13.p$, $16.p$, $18.p$, $13.q$, $16.q$, and $18.q$. Only statements $5.q$ and $18.q$ may falsify the consequent. As in the proof of (I18), statement $5.p$ may establish the antecedent only if $i/2^{level} = q/2^{level} \wedge (i.j > level \vee i.j = level \wedge i@\{3..14, 17\})$ holds. By $ME(level)$, this implies that the antecedent does not hold. The reasoning for statements $6.p$, $13.p$, $16.p$, $18.p$, $13.q$, and $16.q$ is the same as in the proof of (I18). Statement $18.q$ may falsify (I34) only when $p.j = level \;\wedge\; (p@\{6\} \;\wedge\; p.rival \neq -1 \;\vee\; p@\{7..12\}) \;\wedge\; q/2^{level} = comp(p/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{18\}$ holds. By (I31), this implies that $q.j = level \;\wedge\; T[level, q/2^{level+1}] = p$ holds. By (I3), this implies that statement $18.q$ establishes the consequent, and hence preserves (I34). Finally, although statement $5.q$ may falsify the consequent, it establishes $q@\{6\}$. □

invariant $p.j = level \;\wedge\; p@\{11,12\} \;\wedge\; (\forall i : i/2^{level} = comp(p/2^{level}) :: i.j < level \;\vee$
$i.j = level \;\wedge\; i@\{0..3, 15, 16\}) \;\Rightarrow\; P[level,p] = 2$ (I35)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I35) is true. The antecedent may be established only by statements $10.p$, $13.p$, $16.p$, $13.i$, $14.i$, $15.i$, $16.i$, $19.i$, and $20.i$. Only statements $4.p$ and $9.i$ may falsify the consequent. Statement $10.p$ may establish the antecedent only if $p.j = level \;\wedge\; p@\{10\} \;\wedge\; (\forall i : i/2^{level} = comp(p/2^{level}) :: i.j < level \;\vee\; i.j = level \;\wedge\; i@\{0..3, 15, 16\}) \;\wedge\; P[level,p] \geq 1$ holds. By (I33), this implies that $(P[level,p] = 0 \;\vee\; P[level,p] = 2) \;\wedge\; P[level,p] \;\geq 1$ holds. This implies that whenever statement $10.p$ establishes the antecedent, $P[level,p] = 2$ holds. $\neg p@\{11,12\}$ holds after the execution of statements $13.p$ and $16.p$.

The reasoning for statements $13.i$, $14.i$, $15.i$, and $16.i$ is the same as given in the proof of (I33). Statements $19.i$ and $20.i$ establish the antecedent only if $p.j = level \;\wedge\; p@\{11,12\} \;\wedge\; i/2^{level} = comp(p/2^{level}) \;\wedge\; i.j = level \;\wedge\; i@\{19,20\}$ holds. By (I34), this implies that $i.j = level \;\wedge\; i@\{19,20\} \;\wedge\; i.rival = p$ holds. In that case, statement $19.i$ establishes $i@\{20\}$, and statement $20.i$ establishes $P[level,p] = 2$.

Although statement $4.p$ may falsify the consequent, it establishes $p@\{5\}$. (I9) implies that statement $9.i$

may falsify the consequent only if $p/2^{level} = comp(i/2^{level}) \wedge i.j = level \wedge i@\{9\}$ holds. By (C1), this implies that $i/2^{level} = comp(p/2^{level}) \wedge i.j = level \wedge i@\{9\}$ holds, which implies that the antecedent does not hold. $\Box$

invariant $p.j = level \wedge p@\{5..12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{9\} \wedge q.rival = p \Rightarrow P[level, p] = 0$ (I36)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I36) is true. The antecedent may be established only by statements $4.p$, $13.p$, $16.p$, $5.q$, $8.q$, $13.q$, $16.q$, and $18.q$. Only statements $9.i$ and $20.i$ may falsify the consequent. Statement $4.p$ may establish the antecedent only if $p.j = level$ holds. In this case, it also establishes $P[level, p] = 0$. $\neg p@\{5..12\}$ holds after the execution of statements $13.p$ and $16.p$. $\neg q@\{9\}$ holds after the execution of statements $5.q$, $13.q$, $16.q$, and $18.q$. Statement $8.q$ establishes the antecedent only when $q.j = level \wedge q@\{8\} \wedge q.rival = p \wedge P[level, q.rival] = 0$ holds. This implies that the consequent holds.

(I9) implies that statement $9.i$ may falsify (I36) only if $i.j = level \wedge i@\{9\} \wedge q/2^{level} = comp(p/2^{level}) \wedge p/2^{level} = comp(i/2^{level})$ holds. By (C1), this implies that $i.j = level \wedge i@\{9\} \wedge q/2^{level} = i/2^{level}$ holds. If $i = q$, then statement $9.i$ establishes $q@\{10\}$. If $i \neq q$, then $ME(level)$ implies that $q.j = level$ does not hold. Thus, statement $9.i$ preserves (I36). (I11) implies that statement $20.i$ may falsify (I36) only if $q/2^{level} = comp(p/2^{level}) \wedge p/2^{level+1} = i/2^{level+1} \wedge i.j = level \wedge i@\{20\}$ holds. By (C1), this implies that $comp(q/2^{level}) = p/2^{level} \wedge q/2^{level} = comp(p/2^{level}) \wedge p/2^{level+1} = i/2^{level+1} \wedge i.j = level \wedge i@\{20\}$ holds. By (C2), this implies that $(i/2^{level} = p/2^{level} \vee i/2^{level} = q/2^{level}) \wedge i.j = level \wedge i@\{20\}$ holds. If $i = p$, then statement $20.i$ establishes $p@\{15\}$. If $i/2^{level} = p/2^{level} \wedge i \neq p \wedge i.j = level \wedge i@\{20\}$ holds, then $ME(level)$ implies that $p.j = level$ does not hold. If $i = q$, then statement $20.i$ establishes $q@\{15\}$. If $i/2^{level} = q/2^{level} \wedge i \neq q \wedge i.j = level \wedge i@\{20\}$ holds, then $ME(level)$ implies that $q.j = level$ does not hold. $\Box$

invariant $p.j = level \wedge p@\{12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{4..10\} \wedge T[level, p/2^{level+1}] = q \Rightarrow P[level, p] = 2$ (I37)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I37) is true. The antecedent may be established only by statements $11.p$, $13.p$, $16.p$, $3.q$, $13.q$, and $16.q$. Only statements $4.p$ and $9.i$ may falsify the consequent. (I3) implies that statement $11.p$ may establish the antecedent only if $p.j = level \wedge p@\{11\} \wedge T[level, p/2^{level+1}] = p$ holds. Thus, statement $11.p$ does not establish the antecedent. $\neg p@\{12\}$ holds after the execution of statements $13.p$ and $16.p$.

Statement $3.q$ establishes the antecedent only if $p.j = level \wedge p@\{12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{3\}$ holds. By $ME(level)$, this implies that $p.j = level \wedge p@\{12\} \wedge q/2^{level} = comp(p/2^{level}) \wedge q.j = level \wedge q@\{3\} \wedge (\forall i : i/2^{level} = comp(p/2^{level}) \wedge i \neq q :: i.j < level)$ holds. In that case, (I35) implies that $P[level, p] = 2$ holds. $\neg q@\{4..10\}$ holds after the execution of statements $13.q$ and $16.q$.

Although statement $4.p$ may falsify the consequent, it establishes $p@\{5\}$. As in the proof of (I36), statement $9.i$ may falsify (I37) only if $i/2^{level} = q/2^{level} \wedge i.j = level \wedge i@\{9\}$ holds. If $i = q$, then (I36) implies that statement $9.i$ does not falsify the consequent when the antecedent holds. If $i \neq q$, then $ME(level)$ implies that $q.j = level$ does not hold. $\Box$

Now, we prove that the algorithm is free from starvation. To facilitate the presentation, we define the following predicate.

$$SF(level) \equiv (\forall i :: i.j = level \ \wedge\ i@\{1..13\} \ \mapsto\ i@\{14\})$$

We next prove that the following assertion holds, which implies that the starvation-freedom property holds.

$$(\forall n : 0 \le n \le \log_2 N :: SF(n)) \tag{G2}$$

We use an induction on n in the proof. Observe that $SF(\log_2 N) \equiv (\forall i :: i.j = \log_2 N \ \wedge\ i@\{1..13\} \ \mapsto\ i@\{14\})$. By the contrapositive of (I1), $(\forall i :: i.j = \log_2 N \ \Rightarrow\ \neg i@\{2..13\})$ holds. By the definition of a fair history and the program text, $(\forall i :: i.j = \log_2 N \ \wedge\ i@\{1\} \ \mapsto\ i@\{14\})$ holds. These two assertions imply that the induction base $SF(\log_2 N)$ holds. Thus, it suffices to prove that the following assertion holds.

$$(\forall level : 0 \le level < \log_2 N :: (\forall j : j > level :: SF(j)) \ \Rightarrow\ SF(level)) \tag{G3}$$

Next, we prove several assertions that are needed to establish $SF(level)$. In these proofs, we assume that $(\forall j : j > level :: SF(j))$ holds.

$$i.j > level \ \mapsto\ i.j = level \ \wedge\ i@\{19\} \tag{L1}$$

By (I0), $i.j > level \ \Rightarrow\ i@\{1..20\}$. By our assumption that $(\forall j : j > level :: SF(j))$ holds, by (I1), $i.j > level \ \wedge\ i@\{1..13\} \ \mapsto\ i.j > level \ \wedge\ i@\{14\}$. By the definition of a fair history and the program text, $i.j > level \ \wedge\ i@\{14..20\} \ \mapsto\ i.j = level \ \wedge\ i@\{19\}$. □

Now, we prove two *unless* assertions.

$$i.j = level \ \wedge\ i@\{10\} \ \wedge\ P[level, i] \ge 1 \ \ \textit{unless}\ \ i.j = level \ \wedge\ i@\{11\} \tag{U0}$$

To prove that (U0) holds, it suffices to consider only those statements that may falsify $i.j = level \wedge i@\{10\} \wedge P[level, i] \ge 1$. The statements to consider are $4.i$, $10.i$, $13.i$, and $16.i$. When statements $4.i$, $13.i$, or $16.i$ are enabled for execution, $\neg i@\{10\}$ holds, which implies that these statements preserve (U0). Statement $10.i$ establishes $i.j = level \ \wedge\ i@\{11\}$, if executed when $i.j = level$ holds, and hence preserves (U0).

$$i.j = level \ \wedge\ i@\{12\} \ \wedge\ P[level, i] = 2 \ \ \textit{unless}\ \ i.j = level \ \wedge\ i@\{13\} \tag{U1}$$

To prove that (U1) holds, it suffices to consider only those statements that may falsify $i.j = level \wedge i@\{12\} \wedge P[level, i] = 2$. The statements to consider are $4.i$, $12.i$, $13.i$, $16.i$, and $9.q$. When statements $4.i$, $13.i$, or $16.i$ are enabled for execution, $\neg i@\{12\}$ holds, which implies that these statements preserve (U1). Statement $12.i$ establishes $i.j = level \ \wedge\ i@\{13\}$, if executed when $i.j = level$ holds, and hence preserves (U1). (I9) implies that statement $9.q$ may falsify (U1) only when $i.j = level \ \wedge\ i@\{12\} \ \wedge\ q.j = level \ \wedge\ q@\{9\} \ \wedge\ i/2^{level} = comp(q/2^{level})$ holds. By (C1), this implies that $i.j = level \ \wedge\ i@\{12\} \ \wedge\ q.j = level \ \wedge\ q@\{9\} \ \wedge\ q/2^{level} = comp(i/2^{level})$ holds. In this case, (I36) implies that $q.rival \ne i \ \vee\ P[level, i] = 0$ holds, which implies that statement $9.q$ does not falsify (U1).

The next two assertions follow from these *unless* assertions, the definition of a fair history, and the program text; (U0) is used to prove (L2) and (U1) is used to prove (L3).

$$i.j = level \;\wedge\; i@\{10\} \;\wedge\; P[level, i] \geq 1 \;\mapsto\; i.j = level \;\wedge\; i@\{11\} \tag{L2}$$

$$i.j = level \;\wedge\; i@\{12\} \;\wedge\; P[level, i] = 2 \;\mapsto\; i.j = level \;\wedge\; i@\{13\} \tag{L3}$$

The following assertions, which are stated without proof, follow directly from the definition of a fair history and the program text.

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{3..9\} \;\mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{10, 13\}) \end{aligned} \tag{L4}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{11\} \;\mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{12, 13\}) \end{aligned} \tag{L5}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{13\} \;\mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j > level) \end{aligned} \tag{L6}$$

Assertions (L7) through (L13), given next, easily follow from the preceding assertions. In particular, (L1) implies that (L7) holds; (I34) and (L7) imply that (L8) holds; (I34) and (L8) imply that (L9) holds; (L9) implies that (L10) holds; (L2) and (L10) imply that (L11) holds; (I27) implies that (L12) holds; and (L2) implies that (L13) holds.

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; (q.j > level \;\vee\; q.j = level \;\wedge\; q@\{17..19\}) \mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{19\}) \end{aligned} \tag{L7}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; (q.j > level \;\vee\; q.j = level \;\wedge\; q@\{17..19\}) \mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{19\} \;\wedge\; q.rival = i) \end{aligned} \tag{L8}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; (q.j > level \;\vee\; q.j = level \;\wedge\; q@\{17..20\}) \mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{20\} \;\wedge\; q.rival = i) \end{aligned} \tag{L9}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; (q.j > level \;\vee\; q.j = level \;\wedge\; q@\{17..20\}) \mapsto \\ & \quad (i.j = level \;\wedge\; i@\{11\}) \vee \\ & \quad (i.j = level \;\wedge\; i@\{10\} \;\wedge\; P[level, i] = 2) \end{aligned} \tag{L10}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; (q.j > level \;\vee\; q.j = level \;\wedge\; q@\{17..20\}) \mapsto \\ & \quad i.j = level \;\wedge\; i@\{11\} \end{aligned} \tag{L11}$$

$$\begin{aligned} & i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{10\} \;\mapsto \\ & \quad i.j = level \;\wedge\; i@\{10\} \;\wedge\; q/2^{level} = comp(i/2^{level}) \;\wedge\; q.j = level \;\wedge\; q@\{10\} \;\wedge\; (P[level, i] \geq 1 \;\vee \end{aligned}$$

$P[level,q] \geq 1)$ (L12)

$i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{10\} \ \wedge \ (P[level,i] \geq 1 \vee P[level,q] \geq 1) \mapsto$
$(i.j = level \ \wedge \ i@\{11\}) \vee$
$(i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{11\})$ (L13)

invariant $i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \Rightarrow$
$(i.j = level \ \wedge \ i@\{10\} \ \wedge \ P[level,i] \geq 1) \vee$
$(i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \wedge \ P[level,q] = 2)$ (I38)

By (I16), the antecedent implies that $i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \wedge \ (T[level, i/2^{level+1}] = q \ \vee \ T[level, i/2^{level+1}] = i)$ holds. By (C1) and (C3), this implies that $(i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \wedge \ T[level, q/2^{level+1}] = q) \ \vee \ (i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ i/2^{level} = comp(q/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \wedge \ T[level, q/2^{level+1}] = i)$ holds. By (I26) and (I37), this implies that the consequent of (I38) holds. □

$i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{12\} \ \mapsto$
$(i.j = level \ \wedge \ i@\{11\}) \vee$
$(i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{13\})$ (L14)

By (L2) and (L3), (I38) implies that (L14) holds. □

$i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{3..14\} \ \mapsto$
$(i.j = level \ \wedge \ i@\{11\}) \vee$
$(i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j > level)$ (L15)

(I2) implies that $\neg(q.j = level \ \wedge \ q@\{14\})$ holds. (Recall that $level < \log_2 N$.) Hence, (L4), (L5), (L6), (L12), (L13), (L14), and $\neg(q.j = level \ \wedge \ q@\{14\})$ imply that (L15) holds. □

$i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{3..14\} \ \mapsto$
$(i.j = level \ \wedge \ i@\{11\})$ (L16)

(L11) and (L15) imply that (L16) holds. □

$i.j = level \ \wedge \ i@\{10\} \ \mapsto \ i.j = level \ \wedge \ i@\{11\}$ (L17)

(I24) implies that $i.j = level \ \wedge \ i@\{10\} \ \Rightarrow \ (i.j = level \ \wedge \ i@\{10\} \ \wedge \ P[level,i] = 2) \ \vee \ (\exists q :: i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ (q.j > level \ \vee \ q.j = level \ \wedge \ q@\{17..20\})) \ \vee \ (\exists q :: i.j = level \ \wedge \ i@\{10\} \ \wedge \ q/2^{level} = comp(i/2^{level}) \ \wedge \ q.j = level \ \wedge \ q@\{3..14\})$. By (L2), (L11), and (L16), this implies that (L17) holds. □

By proving assertions similar to (I38) and (L4) through (L16), it is possible to establish (L18), given next, which is similar to (L17). For brevity, we omit the proof of (L18).

$$i.j = level \ \wedge \ i@\{12\} \ \mapsto \ i.j = level \ \wedge \ i@\{13\} \tag{L18}$$

Note that (L17) implies that the first busy-waiting loop of process i terminates, while (L18) implies that the second busy-waiting loop of i terminates.

$$i.j = level \ \wedge \ i@\{1..13\} \ \mapsto \ i.j = level + 1 \ \wedge \ i@\{1\} \tag{L19}$$

By the program text and the definition of a fair history, (L17) and (L18) imply that (L19) holds. (Recall, by assumption, that $level < \log_2 N$. Thus, $i.j = level \ \wedge \ i@\{1\} \ \mapsto \ i.j = level \ \wedge \ i@\{2\}$.) □

Observe that if $(\forall j : j > level :: SF(j))$ holds, then, by (L19), that $SF(level)$ holds. This concludes the proof of (G3). Thus, we conclude that the program in Figure 2 is free from starvation.

5 Fast Mutual Exclusion in the Absence of Contention

As discussed in Section 1, most early mutual exclusion algorithms based on read/write atomicity are neither fast in the absence of contention, nor able to cope with high contention. Because Lamport's fast mutual exclusion algorithm induces $O(1)$ remote operations in the absence of contention, and our mutual exclusion algorithm requires $O(\log_2 N)$ remote operations given any level of contention, it seems reasonable to expect a solution to exist that induces $O(1)$ remote operations when contention is absent, and $O(\log_2 N)$ remote operations when contention is high.

The algorithm given in Figure 3 almost achieves that goal. The basic idea of this modification is to combine Lamport's fast mutual exclusion algorithm and our algorithm, specifically by placing an extra two-process version of our algorithm "on top" of the arbitration tree. The "left" entry section of this extra two-process program (i.e., process u's code in Figure 1) is executed by a process if that process detects no contention. The "right" entry section of this extra program (i.e., process v's code in Figure 1) is executed by the winning process from the arbitration tree. A process will compete within the arbitration tree (as before) if it detects any contention. As seen in Figure 3, the scheme used to detect contention is similar to that used in Lamport's algorithm. In this figure, we use $ENTRY_k$ and $EXIT_k$ to denote the entry and exit sections of the k-process version of our algorithm. A correctness proof for the algorithm of Figure 3 is given in an appendix.

It should be clear that, in the absence of contention, a process enters its critical section after executing $O(1)$ remote operations. Also, in the presence of contention, a process enters its critical section after executing $O(\log_2 N)$ remote operations. However, when a period of contention ends, N remote operations might be required in order to re-open the fast entry section — see the **while** loop at line 22 in Figure 3. Nonetheless, performance studies show that, under high contention, these statements are rarely executed. (Under low contention, they are obviously never executed.) For example, out of the 100,000 critical section executions in one experiment, these N statements were performed after only 55 critical section executions in the four-process case, and after only one in the eight- and sixteen-process cases.

In the absence of contention, our algorithm generates about twice as many remote memory operations as Lamport's. However, under high contention, our algorithm is clearly superior, as Lamport's induces an unbounded number of remote operations. Also, our modified algorithm ensures starvation-freedom, whereas Lamport's algorithm does not.

```
shared var B : array[0..N − 1] of boolean;
           X, Y : −1..N − 1;
           Z : boolean
initially  Y = −1 ∧ Z = false ∧ (∀i :: B[i] = false)

process i
private var flag : boolean;
            n : 1..N

while true do
  TOP:     0: Noncritical Section;
           1: X := i;
           2: if Y ≠ −1 then goto SLOW fi;
           3: Y := i;
           4: if X ≠ i then goto SLOW fi;
           5: B[i] := true;
           6: if Z then goto SLOW fi;
           7: if Y ≠ i then goto SLOW fi;
           8: ENTRY_2;                           /* Two-Process Entry Section */
           9:   Critical Section;
          10: EXIT_2;                            /* Two-Process Exit Section */
          11: Y := −1;
          12: B[i] := false;
          13: goto TOP;

  SLOW:   14: ENTRY_N;                           /* Arbitration Tree */
          15:   ENTRY_2;                         /* Two-Process Entry Section */
          16:     Critical Section;
          17:     B[i] := false;
          18:     if X = i then
          19:       Z := true;
          20:       flag := true;
          21:       n := 0;
          22:       while (n < N) do
          23:         if B[n] then flag := false fi;
          24:         n := n + 1
                    od;
          25:       if flag then Y := −1 fi;
          26:       Z := false
                  fi;
          27:   EXIT_2;                          /* Two-Process Exit Section */
          28: EXIT_N                             /* Arbitration Tree */
od
```

Figure 3: Fast, scalable mutual exclusion algorithm.

6 Performance Results

To compare the scalability of our mutual exclusion algorithm with that of other algorithms, we conducted a number of experiments on the BBN TC2000 and Sequent Symmetry multiprocessors. Results from some of these experiments are presented in this section.

BBN TC2000

The BBN TC2000 is a distributed shared memory multiprocessor, each node of which contains a processor and a memory unit. The nodes are connected via a multi-stage interconnection network, known as the Butterfly switch. Each access to a remote memory location (i.e., one that requires a traversal of the interconnection network) takes about 2 microseconds, whereas each local reference takes about 0.6 microseconds. Each node's processor, a Motorola 88100, provides an atomic fetch-and-store instruction called `xmem`. Other strong primitives such as compare-and-swap and fetch-and-add are provided using the TC2000 hardware locking protocol [4]. The TC2000 has cache memory, but does not provide a cache coherence mechanism.

We tested seven mutual exclusion algorithms on the TC2000: a simple test-and-set algorithm; the queue-based algorithm using compare-and-swap given by Mellor-Crummey and Scott in [10]; the queue-based algorithm using fetch-and-add given by T. Anderson in [3]; the fast mutual exclusion algorithm given by Lamport in [9]; the tree-based algorithm given by Styer in [13]; the tree-based algorithm given by Peterson and Fischer in [12]; and the mutual exclusion algorithm described in Section 3. Performance results obtained by running these seven algorithms on the TC2000 are summarized in Figure 4. Each point (x, y) in each graph represents the average time y for one critical section execution with x competing processors. The timing results summarized in the graph were obtained by averaging over 10^5 critical section executions. The critical section consists of a read and an increment of shared counter. Results obtained using larger critical sections, which for brevity are not presented here, show similar performance to that depicted in Figure 4. The timing results presented include the execution time of critical sections.

The performance of the test-and-set algorithm is given by the graph labeled T&S, Mellor-Crummey and Scott's algorithm by the graph labeled MCS, T. Anderson's algorithm by the graph labeled AND, Lamport's algorithm by the graph labeled LAMP, Styer's algorithm by the graph labeled STYER, Peterson and Fischer's algorithm by the graph labeled PF, and our algorithm by the graph labeled YA. On the TC2000, the MCS algorithm was the best overall performer of the alternatives considered here. The graph depicted for the MCS algorithm is mostly flat, except at the point for two processors. This anomaly at two processors coincides with results reported by Mellor-Crummey and Scott on the Sequent Symmetry, and was attributed by them to the lack of a compare-and-swap instruction on the Symmetry [10]. As our implementation of their algorithm did employ compare-and-swap, we have not found a satisfying explanation for this behavior on the TC2000.

T. Anderson's algorithm requires only local spinning when implemented on a machine with coherent caches. On the Symmetry, where each process can spin on its own coherent cache, Anderson's algorithm outperforms the MCS algorithm. However, on the TC2000, which does not support coherent caching, Anderson's algorithm requires remote spinning, slowing its performance.

The simple T&S algorithm exhibited poor scalability. The average execution time for the 64 processor case, which is not depicted in Figure 4, is about 330 microseconds. Where there is a possibility of contention among a large number of processors, it should be avoided, or used with good backoff scheme [1].

Three algorithms based on atomic reads and writes — Lamport's, Peterson and Fischer's, and Styer's — also showed poor scalability. In particular, the performance of Lamport's algorithm degrades dramatically as the number of contenders increases. The average execution time for the 64 processor case, which is not

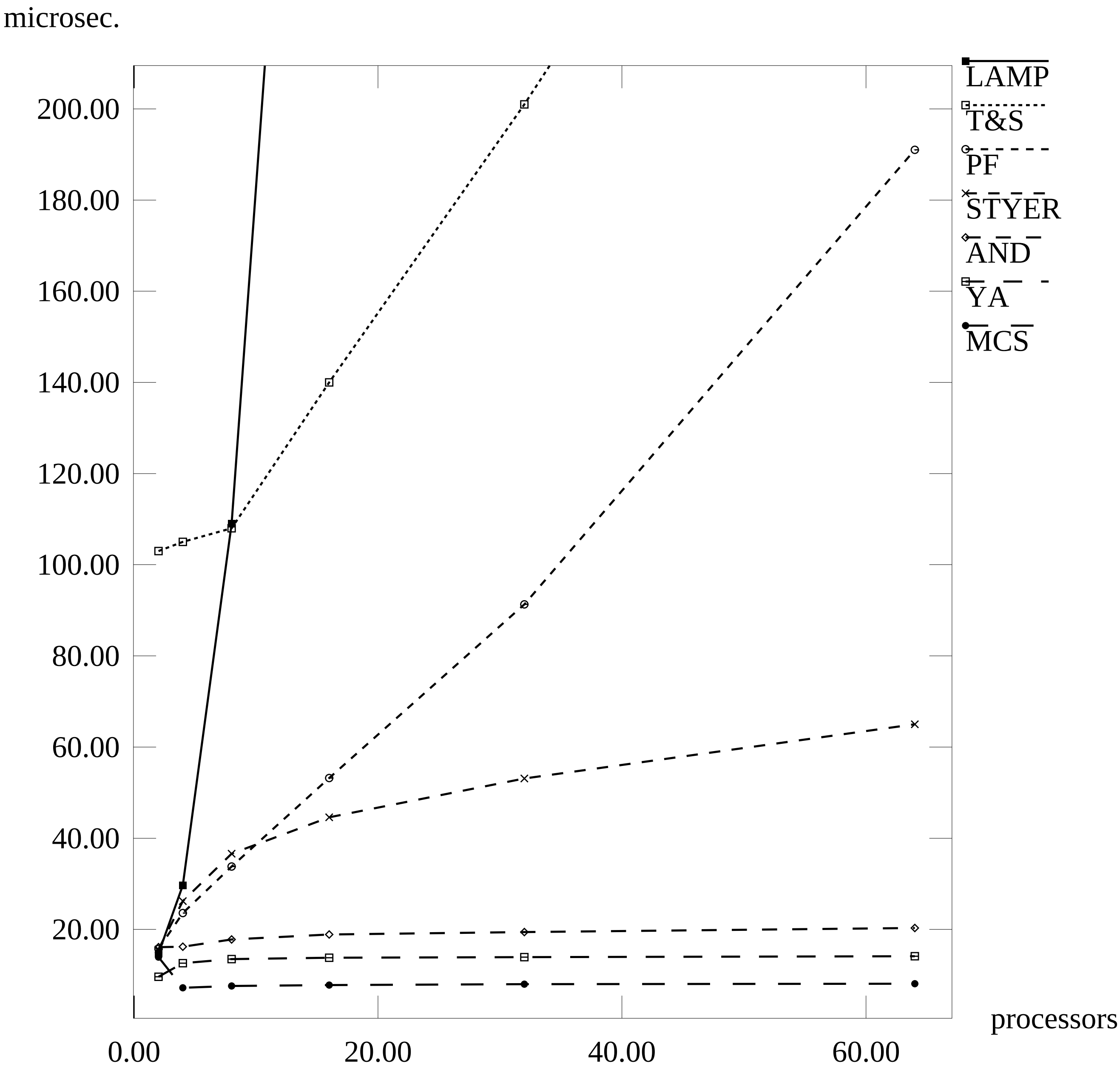

Figure 4: TC2000, small critical section.

depicted in Figure 4, is about 4000 microseconds. The performance of Styer's algorithm, which is better than that of Lamport's, is due to the tree structure employed. Styer's algorithm generates $O(\log_2 N)$ remote operations outside of busy-waiting loops. Even though Peterson and Fischer's algorithm is also tree-based, it induces $O(N)$ remote operations outside of busy-waiting loops, which results in poorer scalability.

Our mutual exclusion algorithm shows performance that is comparable to that of T. Anderson's and Mellor-Crummey and Scott's algorithms. Its good scalability emphasizes the importance of local spinning. The difference seen between our mutual exclusion algorithm and the MCS algorithm is explained by the amount of global traffic generated by each algorithm. The MCS algorithm generates $O(1)$ remote operations per critical section execution, whereas ours generates $O(\log_2 N)$. The global traffic of the other five algorithms is unbounded, as each employs global spinning. The performance of T. Anderson's algorithm is far better than that of the simple test-and-set algorithm. Because the processes in Anderson's algorithm spin globally on the TC2000, this might be interpreted as a counterexample to our belief that minimizing remote operations is important for good scalability. However, Mellor-Crummey and Scott reported in [10] that Anderson's algorithm produced far fewer remote operations than the test-and-set algorithm.

Sequent Symmetry

Performance results of experiments on the Sequent Symmetry are summarized in Figure 4. The Sequent Symmetry is a shared memory multiprocessor whose processor and memory nodes are interconnected via a shared bus. A processor node consists of an Intel 80386 and a 64 Kbyte, two-way set-associative cache. Cache coherence is maintained by a snoopy protocol. The Symmetry provides an atomic fetch-and-store instruction. Because other strong primitives are not provided, we used a version of Mellor-Crummey and Scott's algorithm that is implemented with fetch-and-store and that does not ensure starvation-freedom [10]. Fetch-and-add, which is used in T. Anderson's algorithm, was simulated by a test-and-set algorithm with randomized backoff, as Anderson did in [3].

The experiments on the Symmetry show similar results to that for the TC2000. However, on the Symmetry, T. Anderson's algorithm has the best overall performance, mainly because the availability of coherent caches makes all spins in his algorithm local. The performance of Lamport's algorithm on the Symmetry is far better than that on the TC2000. This seems partly due to the fact that his algorithm is not starvation-free. Specifically, when a process enters its critical section, it can keep all needed variables in its own cache and repeatedly enter its critical section, without yielding to the other processes. In one of our tests for the two-process case, one process executed 50,000 critical sections during a period of time in which the other process executed only 120 critical sections.

Dependence on coherent caching for efficient synchronization [3, 7] is questionable, as many caching schemes do not cache shared writable data. Our solution neither requires a coherent cache for efficient implementation nor any strong primitives. An efficient implementation of our algorithm requires only that each processor has some part of shared memory that is locally accessible, and that read and write operations are atomic. We consider these to be minimal hardware requirements for efficient synchronization. It is worth noting that, without fetch-and-add and compare-and-swap primitives, T. Anderson's algorithm and Mellor-Crummey and Scott's algorithm are not starvation-free.

7 Concluding Remarks

We have presented a scalable mutual exclusion algorithm for shared memory multiprocessors that does not require any hardware support other than atomic read and write operations. Our algorithm has better

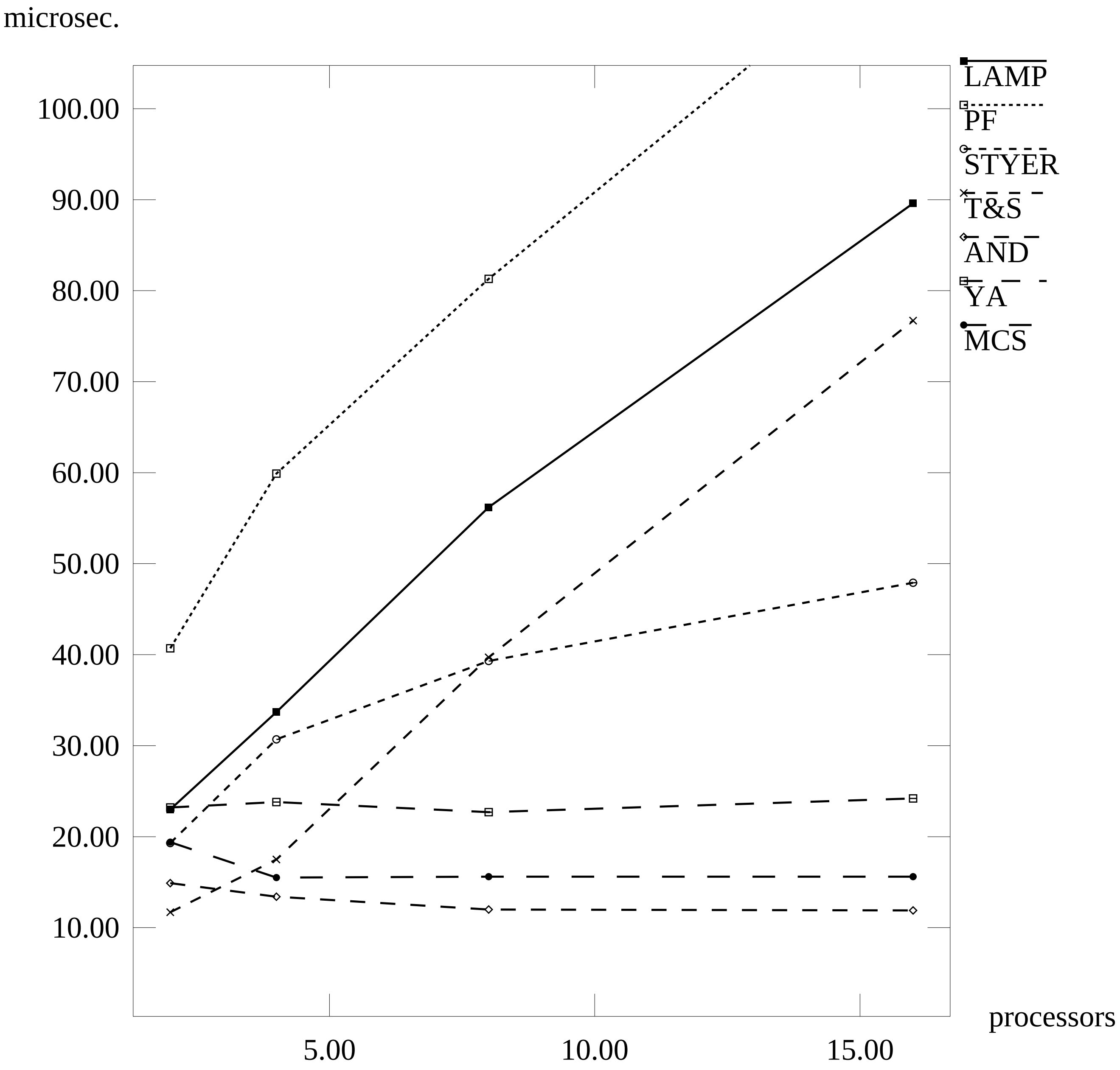

Figure 5: Symmetry, small critical section.

worst-case time complexity than any previously published mutual exclusion algorithm based on read/write atomicity, requiring $O(\log_2 N)$ remote operations under any amount of contention. We have also presented an extension of our algorithm for fast mutual exclusion in the absence of contention that generates $O(1)$ remote operations in the presence of contention and $O(N)$ in the absence of contention. In addition, we have suggested two open problems concerning the power of atomic reads and writes as synchronization primitives.

In the complexity calculations given in this paper, the distinction between remote and local operations is based upon a static assignment of shared variables to processes. Other definitions, which incorporate specific architectural details of systems, are also possible. For example, for programs intended for machines with coherent caching, it might be appropriate to consider a read of a shared variable x by a process p to be local if x has not been written by another process since p's most recent access of x. However, because of the many parameters that go into defining a cache-coherence protocol, such definitions can be problematic. We therefore choose to leave this as a subject for further study.

A natural approach to measuring the time complexity of concurrent programs would be to simply count the total number of operations executed. However, a straightforward application of such an approach does not provide any insight into the behavior of mutual exclusion algorithms under heavy contention. In particular, in any algorithm in which processes busy-wait, the number of operations needed for one process to get to its critical section is unbounded. In order to serve as a measure of time complexity, a measure should be both intuitive and easy to compute. In sequential programming, the usual measure of time complexity, which is obtained by simply counting operations, satisfies these criteria. By contrast, there has been much disagreement on how time complexity should be measured in concurrent programs, and a complexity measure satisfying these criteria has yet to be adopted. We believe that an appropriate time complexity measure for concurrent algorithms is one based on the number of remote memory references. As seen in this paper, such a measure can be used to make meaningful distinctions concerning the performance of concurrent programs.

Acknowledgement: We would like to thank Howard Gobioff for helping with the performance studies given in Section 6. We would also like to acknowledge Argonne National Laboratories and Lawrence Livermore National Laboratories for providing us with access to the machines used in these studies. We are particularly grateful to Terry Gaasterland at Argonne, and Tammy Welcome and Brent Gorda at Lawrence Livermore for their help.

Appendix: Correctness Proof for Fast Algorithm

In this section, we prove that the mutual exclusion and starvation-freedom properties hold for mutual exclusion algorithm of Figure 3. We first prove five invariants that are needed to prove that mutual exclusion holds. The first three are quite simple: (I39) follows from (G0), (I40) follows directly from the program text, and (I41) follows from (I39).

invariant $(\mathbf{N}i :: i@\{15..27\}) \leq 1$ (I39)

invariant $i@\{6..12\} \Rightarrow B[i]$ (I40)

invariant $i@\{20..26\} \Rightarrow Z$ (I41)

invariant $i.flag \wedge ((i@\{22,23\} \wedge i.n > p) \vee (i@\{24\} \wedge i.n \geq p)) \Rightarrow \neg p@\{7..12\}$ (I42)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I42) is true. The antecedent may be established only by statements

20.i, 21.i, 23.i, and 24.i. Only statement 6.p may falsify the consequent. $\neg i@\{22..24\}$ holds after the execution of statement 20.i. Although statement 21.i establishes $i@\{22\}$, it also establishes $i.n = 0$, which implies that the antecedent does not hold. Statement 23.i may falsify (I42) only if $i.flag \wedge i@\{23\} \wedge i.n = p \wedge \neg B[i.n] \wedge p@\{7..12\}$ holds. By (I40), this is a contradiction. Thus, statement 23.i preserves (I42). Statement 24.i may establish $i@\{22\} \wedge i.n > p$ only if $i@\{24\} \wedge i.n \geq p$ holds. Thus, although statement 24.i may preserve the antecedent, it does not establish it. Statement 6.p may falsify the consequent only when Z holds. By (I41), this implies that the antecedent does not hold. $\Box$

invariant $\quad i@\{25\} \wedge i.flag \Rightarrow (\forall p :: \neg p@\{7..12\})$ (I43)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I43) is true. The antecedent may be established only by statements 20.i and 22.i. Only statement 6.q may falsify the consequent. $\neg i@\{25\}$ holds after the execution of statement 20.i. Statement 22.i may establish $i@\{25\} \wedge i.flag$ only if $i@\{22\} \wedge i.flag \wedge i.n \geq N$ holds. By (I42), this implies that the consequent holds. Statement 6.q may falsify the consequent only when Z holds. By (I41), this implies that the antecedent does not hold. $\Box$

The following assertion implies that the mutual exclusion property holds for the fast entry section.

invariant $\quad ((\mathbf{N}i :: i@\{2\} \wedge X = i \wedge Y = -1) + (\mathbf{N}i :: i@\{3\} \wedge X = i) + (\mathbf{N}i :: i@\{4\} \wedge X = i \wedge Y = i) + (\mathbf{N}i :: i@\{5..7\} \wedge Y = i) + (\mathbf{N}i :: i@\{8..11\})) \leq 1 \wedge ((\exists p :: p@\{8..11\}) \Rightarrow Y \neq -1)$ (I44)

Initially $(\forall i :: i@\{0\})$ holds, and hence (I44) is true. (I44) may be falsified only by statements 1.q, 2.q, 3.q, 4.q, 7.q, 11.q, and 25.q.

Statement 2.q may increment $(\mathbf{N}i :: i@\{3\} \wedge X = i)$ only if $q@\{2\} \wedge X = q \wedge Y = -1$ holds. Statement 3.q may increment $(\mathbf{N}i :: i@\{4\} \wedge X = i \wedge Y = i)$ only if $q@\{3\} \wedge X = q$ holds. Statement 4.q may increment $(\mathbf{N}i :: i@\{5..7\} \wedge Y = i)$ only if $q@\{4\} \wedge X = q \wedge Y = q$ holds. Statement 7.q may increment $(\mathbf{N}i :: i@\{8..11\})$ only if $q@\{7\} \wedge Y = q$ holds. Thus, statements 2.q, 3.q, 4.q, and 7.q preserve (I44).

Statement 1.q may falsify (I44) only if $q@\{1\} \wedge Y = -1 \wedge ((\exists p :: p@\{8..11\}) \Rightarrow Y \neq -1)$ holds. This implies that $(\mathbf{N}i :: i@\{4\} \wedge X = i \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{5..7\} \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{8..11\}) = 0$ holds. Because $(\mathbf{N}i :: X = i) \leq 1$ holds, $(\mathbf{N}i :: i@\{2\} \wedge X = i \wedge Y = -1) + (\mathbf{N}i :: i@\{3\} \wedge X = i) \leq 1$ also holds, which implies that statement 1.q preserves (I44).

Statement 7.q establishes $q@\{8\}$ only if $Y = q$ holds, and hence preserves (I44). Statement 11.q could potentially falsify (I44) by establishing $Y = -1$. However, statement 11.q decrements $(\mathbf{N}i :: i@\{8..11\})$ by 1, and hence establishes $(\mathbf{N}i :: i@\{4\} \wedge X = i \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{5..7\} \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{8..11\}) = 0$. Because $(\mathbf{N}i :: X = i) \leq 1$ holds, $(\mathbf{N}i :: i@\{2\} \wedge X = i \wedge Y = -1) + (\mathbf{N}i :: i@\{3\} \wedge X = i) \leq 1$ also holds, which implies that statement 11.q preserves (I44).

Statement 25.q could potentially falsify (I44) by establishing $Y = -1$. However, statement 25.q may establish $Y = -1$ only if $q@\{25\} \wedge q.flag$ holds. By (I43), this implies that $Y = -1 \wedge (\forall i :: \neg i@\{7..12\})$ holds, which implies that statement 25.q establishes $(\mathbf{N}i :: i@\{4\} \wedge X = i \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{5..7\} \wedge Y = i) = 0 \wedge (\mathbf{N}i :: i@\{8..11\}) = 0$. Because $(\mathbf{N}i :: X = i) \leq 1$ holds, $(\mathbf{N}i :: i@\{2\} \wedge X = i \wedge Y = -1) + (\mathbf{N}i :: i@\{3\} \wedge X = i) \leq 1$ also holds, which implies that statement 25.q preserves (I44). $\Box$

ENTRY_2 and EXIT_2 satisfy the following properties.

invariant $\quad ((\mathbf{N}i :: i@\{8..10\}) \leq 1 \wedge (\mathbf{N}i :: i@\{15..27\}) \leq 1) \Rightarrow (\mathbf{N}i :: i@\{9, 16..26\}) \leq 1$ (I45)

$$i@\{8\} \quad \mapsto \quad i@\{9\} \tag{L20}$$

$$i@\{15\} \quad \mapsto \quad i@\{16\} \tag{L21}$$

The proof of (I45) is similar to that of (G1), and is omitted for brevity. The proof of (L20) and (L21) are similar to that of (G3), and are omitted for brevity. (Note that process identifiers and function *comp* are used for convenience in the proof of (G1) and (G3).)

The following two assertions imply that the mutual exclusion and starvation-freedom properties hold for the algorithm of Figure 3.

$$\textbf{invariant} \quad (\mathbf{N}i :: i@\{9,16\}) \leq 1 \tag{I46}$$

(I39), (I44), and (I45) imply that (I46) holds. □

$$\textbf{invariant} \quad i@\{1..8, 14, 15\} \quad \mapsto \quad i@\{9, 16\} \tag{L22}$$

By the program text, $i@\{1..7\} \mapsto i@\{8, 14\}$. (G2) implies that $i@\{14\} \mapsto i@\{15\}$. Hence, (L22) follows from (L20) and (L21). □

References

[1] A. Agarwal and M. Cherian, "Adaptive Backoff Synchronization Techniques", *Proceedings of the 16th International Symposium on Computer Architecture*, May. 1989, pp. 396-406.

[2] J. Anderson, "A Fine-Grained Solution to the Mutual Exclusion Problem", *Acta Informatica*, Vol. 30, No. 3, 1993, pp. 249-265.

[3] T. Anderson, "The Performance of Spin Lock Alternatives for Shared-Memory Multiprocessors", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 1, January, 1990, pp. 6-16.

[4] BBN Advanced Computers, *Inside the TC2000 Computer*, February, 1990.

[5] K. Chandy and J. Misra, *Parallel Program Design: A Foundation*, Addison-Wesley, 1988.

[6] E. Dijkstra, "Solution of a Problem in Concurrent Programming Control", *Communications of the ACM*, Vol. 8, No. 9, 1965, pp. 569.

[7] G. Graunke and S. Thakkar, "Synchronization algorithms for shared-memory multiprocessors", *IEEE Computer*, Vol. 23, June, 1990, pp. 60-69.

[8] J. Kessels, "Arbitration Without Common Modifiable Variables", *Acta Informatica*, Vol. 17, 1982, pp. 135-141.

[9] L. Lamport, "A Fast Mutual Exclusion Algorithm", *ACM Transactions on Computer Systems*, Vol. 5, No. 1, February, 1987, pp. 1-11.

[10] J. Mellor-Crummey and M. Scott, "Algorithms for Scalable Synchronization on Shared-Memory Multiprocessors", *ACM Transactions on Computer Systems*, Vol. 9, No. 1, February, 1991, pp. 21-65.

[11] M. Michael and M. Scott, "Fast Mutual Exclusion, Even With Contention", Technical Report, University of Rochester, June 1993.

[12] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System", *Proceedings of the 9th ACM Symposium on Theory of Computing*, May, 1977, pp. 91-97.

[13] E. Styer, "Improving Fast Mutual Exclusion", *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing*, 1992, pp. 159-168.

[14] J.-H. Yang and J. Anderson, "Fast, Scalable Synchronization with Minimal Hardware Support (Extended Abstract)", *Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, ACM, New York, pp. 171-182, August 1993.

[15] J.-H. Yang and J. Anderson, "On the Time Complexity of Mutual Exclusion and Related Problems", manuscript in preparation, 1993.