# On the Mutual-Exclusion Problem—a quest for minimal solutions.

Uri Abraham and Menachem Magidor
Department of mathematics and computer science
Ben-Gurion University, Be'er Sheva, Israel;
Institute of mathematics
The Hebrew University, Jerusalem, Israel.

November 25, 1999

**Abstract**

We prove that there is no solution with regular registers to the critical-section problem for $n$ processes whose registers are of total size $3n-1$. For $n=2$ this is the best result since for total size six, solutions exist, and are presented here. We emphasize that if one of the two processes never sleeps, then solutions for two processes with regular registers of total size five do exist.

## 1 Preface

The critical-section problem posed by Dijkstra [1965] is the following: Suppose two independent processes that from time to time need to access a common resource (like a printer); how can they coordinate the use of this resource and achieve mutual-exclusion and lockout-freedom? *Mutual-exclusion* prevents the simultaneous use of this resource; and *lockout-freedom* guarantees that whenever a process wishes to access this resource it will be able to do so at some later time. The difficulty of this problem lies in the assumption that there is no supervising process that listens to the processes and tells them when to proceed to the common resource; and likewise, there is no common clock that may provide a coordinating basis for the independent processes. Left to themselves, the processes use a *protocol* (a piece of program) that directs their steps before and after the use of the common resource. These steps include write and read operations to *registers*. (A *register* is a device that carries the values by which the processes communicate.) The question then is to design such a protocol that guarantees mutual-exclusion and lockout freedom.

A rather detailed and formal exposition of the critical-section problem is given in Section 2. This formal exposition is necessary for the proof of our 'impossibility' result: *there is no protocol for the critical-section problem with 2 and 3-valued regular registers* which is presented in Section 4. Whereas for a positive result a presentation of a protocol may suffice, an impossibility result is meaningful only within a clear and precise context which can carry a mathematical proof. We find this context in the notion of 'system-execution' (a version of Lamport [1986]) and protocol-automata.

The nature of the registers used is of fundamental importance. Registers can be classified into multi-writer and single-writer registers: a *multi-writer register* is written and read by all processes, and a *single-writer register* is written by its owner—other processes can only read it. Here we will be interested only in single-writer registers.

In fact, the critical-section problem is a family of distinct problems. When the assumptions on the exact nature of the problem change a little, the problem changes too. For example, Burns et al. [1982], and other authors, investigated the mutual-exclusion problem for $N$ processes that use shared variables that support atomic test-and-set operations. To give another example, Dijkstra [1965] uses common registers that support both read and write operations (Dekker's Protocol). In this paper we consider the model in which, for any process, either reading or writing but not both of a communication variable (a 'register') is possible.

Lamport [1986a] contains a classification of registers (safe, regular and atomic) according to their behavior when a read is concurrent with a write (see Definition 2.4). None of these registers carries a buffer or queue, and any write effaces the previous value.

Again, the critical-section problem changes with the kind of registers used and the number of values they carry (the *size* of the register): S. Huddleston gave a protocol for two processes that use a 2-valued atomic register and a regular 3-valued register (unpublished). Peterson and Fischer [1977] have a solution that uses three values in each of the two atomic registers (their solution is symmetric for the two processes). Both of these protocols fail if the registers are only assumed to be regular. For regular registers, Peterson [1983] gave a solution for any number of processes using a 4-valued register each. The 2-4 protocol for regular registers given in Section 5 was known long ago to Lamport and to Huddleston, but it was never published.

Another factor determining the nature of the problem is whether the processes are required to be active forever (accessing again and again the common resource) or are allowed to stay from some moment on in an external inactive state. Our impossibility result refers to protocols that allow a possible non-ending external state. In case one of the processes is active forever, protocols with regular registers of sizes 2 and 3 do exist (and are given here).

In Lamport [1986] the critical-section problem is presented with registers that are boolean (two-valued) and regular; but the number of registers per process is not limited. In our paper we are assuming that each process possesses a **single** register and try to minimize the size of the registers used.

In their [1982] paper, Burns, Jackson, Lynch, Fischer, and Peterson address the same questions we do—seeking minimal solutions to the mutual-exclusion problem—but their assumption is that the processes use a single shared variable instead of registers. This variable can be approached by any process for an atomic test-and-set operation. Thus they deal with a different member of the mutual-exclusion problem family, and they ask what bounds are obtainable for a model in which only reading or writing of a variable, but not a combination of the two, is indivisible? We give here a solution to this question. (In the case of more than two processes an open question still remains—see the paper's last paragraph.)

Following is a description of the results of the paper. Section 4 contains the main result: we show that there is no solution with regular registers to the critical-section problem when the total size of the registers used is five. This easily implies the general result for $n$ processes: that the sum of the sizes of the regular registers is $\geq 3n$ (see Lemma 2.5). In the impossibility proof, we use protocol-automata and these are described in the Introduction. The Introduction also contains a formal definition of the critical-section problem as well as a definition of safe, regular, and atomic registers.

We present mainly two positive results: critical-section protocols for regular registers with a total size of six values. The first protocol is for registers of sizes two and four (Section 5.1); and the second protocol is for two 3-valued registers (Section 5.2). To our surprise, if one assumes that one of the two processes never sleeps, then solution with a total register size of five exist. A variant of the protocol of Section 5.2 gives a protocol for registers that are 2 and 3-valued provided that the writer of the 2-valued register is not permitted to remain eternally in its external state; and a variant of the protocol of Section 5.1 deals with the case where the writer of the 3-valued register may not remain in its external state eternally.

Section 3 deals with the case of two boolean registers. This case is easier than the main problem of 2 and 3 values, but surprisingly it is not totally trivial. It is easy to prove that there is no solution to the critical-section problem for two processes if each has one boolean regular register, even if both processes never sleep. In Section 3 we prove that this impossibility result is still true even if only one boolean register is assumed to be regular and the other boolean register is atomic (this does not follow from our main impossibility result which does not rule out protocols that are active forever).

We believe that our method of proof is as important as the result itself. Perhaps this is the first impossibility result that goes so deep in analyzing the automata involved. Compared to the elegant Impossibility of Distributed Consensus result of Fisher, Lynch and Paterson [1983], for example, we need to look at the automata themselves and not merely at the set of possible scenarios. This makes our proof harder. For infinite protocols, there is a need in our case to resort to a special diagonalizing argument. The argument is new in this context, and may be necessary in other results that deal with infinite protocols.

# 2 Introduction and preliminaries

This section describes the framework within which our results are obtained, defines what is an automaton, and gives a version of the notion of *system-execution* (L. Lamport [1986a] tailor-made to our specific needs). We assume global-time. This is not a limitation of our protocols, but a convenient assumption and later we will say more about this.

Let us begin with an intuitive description of automata. Finite, deterministic, communicating automata represent protocols. The passage from one node to the next is determined by a communication event: either a write or a read (so internal computational events are not explicitly represented in this model). In the automaton model, the events (operation-executions) are represented by edges leading from one node to another. A write node is tagged with a register name, and a single, value-tagged edge is leaving it. This edge represents the event of writing the tagged value on the register. A read node indicates which register is to be read. Since the outcome of the read is determined by factors outside the protocol, there is one tagged edge leaving the read node for each possible value. Each read edge represents the reading event that obtained its value. Other types of nodes are needed too: critical-section, and external nodes. A single edge leaves a critical-section or an external node; such edges are called critical-section edges and external edges respectively. A critical-section node represents a state of the process that is about to start its critical activity, and this activity is represented by the critical-section edge leaving it. The external-state edge represents all those activities of the process that are out of its critical-section; if and when these external activities are terminated the process starts to execute the protocol proper. From the point of view of the protocol, the external edge represents the event of being dormant. The external event may last forever, but any other event has a bounded duration. It is clear that protocols that are given as programs can be transformed into protocol automata, so our results apply to the general notion of protocols. A node of the automaton corresponds to a state in the computation, that is to the program's variables values and to the control point (the place of the next statement to be executed).

We consider infinite protocol automata to allow for protocols that evolve in time. For instance, the protocol may change its behavior depending on the history of the previous executions by that process. Considering infinite automata makes our impossibility result stronger.

We assume that the value each register carries is from a finite number of data values—this number is the *size* of the register. The registers are assumed to be single-writer.

Now comes the formal definition (the parenthetical explanatory remarks are not part of it).

**Definition 2.1** *A* protocol automaton *is a graph consisting of tagged nodes and (directed) edges. (The nodes represent states, and the edges represent events.) There are four kinds of nodes and edges: write, read, external and critical-section.*

1. *A* write node *is tagged by a register name and a single edge leaves it. This edge is tagged with a value, $v$, and is called a 'write-of-$v$' edge.*

2. *A* read node *has a register name attached to it, and if $V$ is the set of possible values carried by that register, then for each $v \in V$ there is an edge tagged with $v$ leaving the node. This edge is called a 'read-of-$v$' edge. Only read edges leave a read node, and only one read-of-$v$ edge for each value $v$.*

3. *A single edge leaves an external node. (It represents the activity of the process that is external to the protocol. The process can stay forever in this external state.) There can be several external nodes and one distinguished external node is specified to be the 'start node'. The unique edge leaving the start node is called the start edge.*

4. *There is a special write edge, called the initializing write edge, which abuts on the start node (but leaves no node). This edge is tagged with 'initial values' for a list of registers, called 'the registers owned by the automaton' (recall we are assuming single-writer registers). Any register that appears on a write node of the automaton is on this list, but the list may contain other registers as well.*

5. *There is at least one* critical section *node (there can be several critical-section nodes in the automaton). There is a single edge leaving any critical-section node. This critical-section edge represents the activity of the process in the critical-section.*

A *system of automata* is a finite collection, $A_i, i \in I$, of automata which are thought to operate concurrently (the exact meaning of this will be given later). We assume here that any register appearing on a read/write node of one of the automata is owned by a unique $A_i$. When we are assuming only two processes and each one has a single register, a write edge corresponds to the event of writing on the register of its process, and a read edge, to the read of the other process' register. So the write and read nodes need not specify names of registers. Examples of automata can be seen in the figures of Section 5 where a protocol is specified by two automata, one for each process. A write node is depicted by a square and a read node by a circle. A read or write edge is depicted with its value above it. A critical-section node is depicted as an oval with "CS" written on it, and an external node as a square with one heavy side. The initialization edge is tagged with the initial value of the register.

**Definition 2.2** 1. *A* path *in an automaton is a sequence $a_0, a_1, ...,$ of nodes alternating with edges, such that if $a_i$ is an edge then it abuts on node $a_{i+1}$ and emanates node $a_{i-1}$ (for $i > 0$). A path represents a (section of a) possible behavior of the automaton. It can be finite or infinite. (An infinite path is an infinite sequence which may of course have a finite range.)*

*Clearly any path is determined by the sequence of edges in that path, and sometimes we refer to the sequence of edges as the path.*

2. *Let $X$ be a node or an edge in automaton $A$, and $\sigma$ a sequence, $\sigma(1), \sigma(2)...$ of values. The path $P_A(X, \sigma)$ is the unique path that starts with $X$ and has $\sigma(i)$ for the tag value of the i-th read edge after $X$. Since any node other than a read node has a single edge leaving it, this path is well-defined and is determined by $X$ and $\sigma$. (Note that when $X$ is a read edge then $\sigma(1)$ is the value of the* second *read edge in $P_A(X, \sigma)$).*

*When $v$ is a single value, then $P_A(X, v)$ is the path obtained by using the constant function $\sigma(i) = v$. In other words, it is the infinite path obtained starting with $X$ when the register that $A$ is reading has the constant value $v$.*

3. *It is convenient to define the* limited *path $LP_A(X, \sigma)$: It is the longest initial segment of $P_A(X, \sigma)$ that does not contain a write node other than $X$. Thus, if $X$ is the only write node in $P_A(X, \sigma)$, or if that path*

*contains no write nodes, then* $LP_A(X,\sigma) = P_A(X,\sigma)$*; but if* $Y$ *is the first write node in* $P_A(X,\sigma)$ *after* $X$*, then* $LP_A(X,\sigma)$ *is the segment of* $P_A(X,\sigma)$ *from* $X$ *up to (but not including)* $Y$.

When a collection, $A_i$, $i \in I$, of protocol automata are executed, the result is a set of events corresponding to the execution of the protocols' edges. A precedence relation $a \longrightarrow b$ is defined on the events when $a$ precedes $b$. This relation is a partial-order: as the processes work independently of each other, two events of different processes may overlap in time. The events corresponding to a single protocol automaton $A_i$ form a 'process', it is a series of repeated 'executions' of the protocol. Formally the following definition is given. (It is based on Lamport's [1986] notion of system execution and the pomset notion; see Pratt [1986].)

**Definition 2.3** *Given automata,* $A_i, i \in I$*, a* system-execution $\mathcal{S} = (E, \longrightarrow, \alpha)$ *consists of a (finite or countably infinite) set* $E$ *of* events *together with a relation* $\longrightarrow$ *(called the precedence relation on* $E$*) and a partial function* $\alpha$ *on* $E$ *(and in some cases some other functions as well) that satisfy the following:*

1. *The relation* $\longrightarrow$*, defined on* $E$*, is transitive and irreflexive. The relation* $\dashrightarrow$ *is defined by* $a \dashrightarrow b$ *iff* $\neg(b \longrightarrow a)$*. The following properties hold:*

   (a) $a \longrightarrow b \dashrightarrow c \longrightarrow d$ *implies* $a \longrightarrow d$*,*

   (b) *Any* $a \in E$*, is either 'terminating' or 'nonterminating'. If* $a$ *is a terminating event then* $a \longrightarrow b$ *for all* $b \in E$ *except for a finite subset of* $E$*. If* $a$ *is a nonterminating event then* $a \longrightarrow b$ *holds for no* $b$*, and the number of* $x$*'s for which* $x \longrightarrow a$ *is finite.*

2. *The correspondence,* $\alpha$*, associates to each event* $e \in E$ *in its domain an edge in one of the automata. We say that* $e$ executes $\alpha(e)$*. (We also say that the event e is associated to the node that* $\alpha(e)$ *is leaving.) If* $\alpha(e)$ *is a read-of-v edge of some register* $\mathsf{R}$*, then* $e$ *is called a 'read of* $\mathsf{R}$*' and we say that* $e$ *returned the value* $v$*. When edge* $\alpha(e)$ *is a write-of-v onto* $\mathsf{R}$*, then* $e$ *is called a 'write' onto* $\mathsf{R}$*, and its 'value' is* $v$*. Similarly, e is called an external, a critical-section or an initializing event depending on whether* $\alpha(e)$ *is an external, a critical-section or an initialization edge. If* $\alpha(e)$ *is the initialization write edge, then e is called the initializing write, it is a write of all the initial values of the registers of its automaton.*

3. *The events assigned by* $\alpha$ *to a single automaton are linearly ordered by* $\longrightarrow$*; these events are called the 'process' of the automaton in the system-execution. The process describes executions of the protocol.*

*That is, the events describe a path in the automaton that begins with the initializing edge. For any initializing event $e$ and read event $r$, $e \longrightarrow r$ holds.*

*We assume that, unless $\alpha(e)$ is an external edge of the automaton, $e$ is a terminating event; if $\alpha(e)$ is an external edge then $e$ can be nonterminating. We assume that the external and critical section events of a process alternate. Between any two critical section events there is an external event, and between any two external events there is a critical section event.*

4. *Each process contains infinitely many events, or else it contains a nonterminating external event. This is the only 'fairness' requirement. When the process does contain infinitely many events then we say that the process 'never sleeps'.*

We now specify the connection between the return value of a read (given by $\alpha$) and the writes onto a register. This connection defines the nature of the register. Intuitively, an atomic register is a serializable one; and a regular register is such that a read that occurs while a writing is still going on returns either the old value or the new one.

**Definition 2.4** *Assume a system-execution $\mathcal{S} = (E, \longrightarrow, \alpha)$.*

1. *Let $r, w \in E$ be a read and a write event of some register* $\mathsf{R}$. *We say that $r$* sees *$w$ iff $w \dashrightarrow r$ and there is no write event $v \neq w$ onto* $\mathsf{R}$ *with $w \longrightarrow v \longrightarrow r$. Since the initializing write, $a$, onto* $\mathsf{R}$ *satisfies $a \longrightarrow r$, there is always at least one write onto* $\mathsf{R}$ *that a read of* $\mathsf{R}$ *can see.*

2. *Register* $\mathsf{R}$ *is called* safe *in the system-execution iff whenever $r$ sees a single write onto* $\mathsf{R}$*, then the value returned is the value written by that single write. We say that this write event* impressed *its value on the read.*

3. *Register* $\mathsf{R}$ *is called* regular *iff the value returned by a read, $r$, is always written by a write that $r$ sees. A function $\Omega$ is called a* regular return function *iff for any read, $r$, of register* $\mathsf{R}$, *$\Omega(r) = w$ for some write, $w$, onto* $\mathsf{R}$ *that $r$ sees, and such that the value written by $w$ is the value returned by $r$. Thus a register is regular in a system-execution iff a regular return function can be defined on its reads.*

4. *A register is called* serial *in the system-execution iff it is safe and the read and write events on* $\mathsf{R}$ *are linearly ordered by $\longrightarrow$.*

5. *Register* $\mathsf{R}$ *is called* atomic *in a system-execution iff in some extension of the system-execution (augmentation of the $\longrightarrow$ relation)* $\mathsf{R}$ *is serial*

*(more on this can be found in Lamport [1986a] where these notions are introduced).*

6. *We say that $\mathcal{S}$ satisfies the* mutual exclusion *property iff any two distinct critical-section events in $\mathcal{S}$ are $\longrightarrow$ comparable ($c_1 \longrightarrow c_2$ or $c_2 \longrightarrow c_1$).*

7. *We say that a process $P$ in system execution $\mathcal{S}$ is lockout free iff either $P$ contains a nonterminating external event, or else it contains infinitely many critical-section events. (In which case every external event is followed by a critical-section event, and every critical section event is followed by an external event. See 3 in Definition 2.3.)*

**Remark**: Given a system execution $\mathcal{S} = (E, \longrightarrow, \alpha)$, a subset $X \subseteq E$ is an *initial segment* iff for every $x \in E$, $e \longrightarrow x$ implies $e \in X$. It is not necessarily the case that every initial segment corresponds to some (initial) execution of the automata. What may go wrong is that for some read $r \in X$, the corresponding write event, $w = \Omega(r)$ is not in $X$ (this can happen if $\neg(w \longrightarrow r)$). This is the reason why we must demand that the initial segments are also closed under the return function $\Omega$. When the initial segment $X$ is also closed under $\Omega$, it is possible to form $\mathcal{S}_1 = (X, \longrightarrow, \alpha \lceil X)$; $\mathcal{S}_1$ is called an initial system execution of $\mathcal{S}$. Assuming that $\Omega$ satisfies $\Omega(x) \dashrightarrow x$ for all $x$'s in the domain of $\Omega$, and the disjointness of the domain and range of $\Omega$, it is not difficult to see that for every event $e$ in $\mathcal{S}$ there is an initial segment closed under $\Omega$ in which $e$ is maximal (there is no $x$ with $e \longrightarrow x$ there).

For given automata and a fixed specification of the registers, the collection of all possible system-executions is called the *system* of these automata (and specification). It describes the set of all possible behaviors. When the registers are specified to be regular, for example, then only system-executions in which the registers are regular are included in the system. Thus, if some property holds in each one of the system-executions in the system, we may claim that the protocol ensures this property provided that the registers are regular.

We say that a system satisfies the *mutual-exclusion property* if any system-execution in it satisfies the mutual-exclusion property.

The system is said to be *lockout free* if each of its processes is lockout free in each system execution in the system.

We are now able to state our theorem:

**Theorem.** *No system for $n$ automata can be lockout free and have the mutual-exclusion property if its registers are specified to be regular and the total size of the registers is $\leq 3n - 1$.*

The proof will be carried in Section 4. Let us now observe that it suffices to prove the Theorem for $n = 2$.

**Lemma 2.5** *Suppose the Theorem above is true for $n = 2$, then it holds for all $n$'s.*

Indeed, assume that we have proved that *No system can be lockout free and have the mutual-exclusion property if its two automata are designed for regular registers of sizes 2 and 3.* Suppose now that there are $n$ automata with total size of registers $\leq 3n - 1$. Then there must be one register with size $< 4$, and another with size $< 3$. But the restriction of the protocol to these two processes contradicts the case $n = 2$.

We would like to say here a few words about the global-time assumption we make here. The *global-time* is the assumption made in defining $a \dashrightarrow b$ as the negation of $b \longrightarrow a$. When it is not made, $\dashrightarrow$ is viewed as an independent relation (this is the approach advocated by Lamport [1986a]). If global-time is assumed, then any system-execution has a representation.

**Definition 2.6** *A function $\mu$ defined on the set of event $E$ is a 'representation' of the system-execution if and only if the following holds: $\mu(e) = [e_1, e_2]$ is an interval (or a single point) in the real line. (It represents the execution time of the event.) For terminating events $e$, $\mu(e)$ is a finite-length interval, and for nonterminating events it is infinite. For all $a, b \in E$, $a \longrightarrow b$ iff $\mu(a)$ is disjoint and to the left of $\mu(b)$. Thus, $\mu$ represents the abstract $\longrightarrow$ relation on $E$.*

Representations can be helpful in motivating the proofs since they allow for pictorial description of the events with intervals. Observe that if we prove, assuming the global-time, that there is no protocol that satisfies a certain property, then a fortiori there is no such protocol at all, since such a protocol would have to work in any system-execution including the global-time ones. For positive results, showing for some protocol that it has some desired properties, it is not immediately clear that the global-time assumption is not a real limitation to the scope of applicability of the protocol. The work by Ben-David [1988] and Abraham, Ben-David and Magidor [1990] shows that, at least for the critical-section problem, global-time is not a limitation. This is the reason why we use this assumption freely, and use pictorial reasoning at some places.

## 2.1 Special features of automata

In many cases it is convenient to assume that our automata have some 'nice' properties which we describe below. These properties may be assumed without loss of generality in the sense that every system of automata may be replaced with one in which the automata satisfy these special properties without increasing the size of the registers, and so that the system with the new automata still satisfies the mutual-exclusion and the lockout freedom properties.

Let us assume that an automaton $\mathcal{A}$ possesses a single register $\mathsf{R}$ on which it writes. It is possible that there are two or more paths leading from the start node to some node $N$, and the value of the register $\mathsf{R}$ at node $N$ depends on whether one path or the other is taken. That is, in one path the last write edge before $N$ has some value $v$, and in the other path the last write edge has a different value. This is somewhat inconvenient, and we would like to assume that every node in the automaton $\mathcal{A}$ corresponds to a state of the process and has a unique register value associated with it. Any automaton can be replaced by an equivalent one satisfying this assumption by splitting and duplicating nodes according to possible values. The system of new automata thus obtained still solves the problem without changing the size of the registers used. We will use the fact that each node has a unique register value in the following manner. Whenever it happens in a system-execution that $r$ is a read event of register $\mathsf{R}$ of $\mathcal{A}$, and for any event $e$ in the process of $\mathcal{A}$, either $e \longrightarrow r$, or $r \longrightarrow e$; then the return-value of $r$ is determined by the edge associated with the rightmost event of $\mathcal{A}$ that precedes $r$, regardless of the history up to that point. Now the following definition makes sense.

**Definition 2.7** *Assume automaton $\mathcal{A}$ writes on a single register* $\mathsf{R}$.

1. *A node $P$ has 'register value' $u$ iff in any path leading from the initializing edge to $P$, the last write edge in that path has value $u$. (The initializing edge is a write edge of the initial value.)*

2. *We say that a path* stabilizes *at value $u$ iff from some point on, all nodes in the path have register value $u$.*

Another natural assumption on the protocol is that it is never required to write a value on the register when that register already has that value. That is, if $W$ is a write-of-$v$ node, then the register value of $W$ is not $v$.

It is often convenient to assume that every node in an automaton is executable. Let us say that an edge or a node $X$ in an automaton is *executable* iff in some system-execution an event is associated with it ($a = \alpha(e)$ for some event $e$ where $a$ is $X$ or is an edge leaving $X$). We can assume that any node in the automaton is executable. To achieve that, simply take only the executable nodes and the executable edges. It is possible that, in doing this, some executable read nodes do not have executable edges for all the possible values (some values are actually never obtained). In that case simply redirect these read edges back to their read nodes.

We have said that *nodes* are executables even though nodes represent states and not actions; intuitively, for a node $X$, '$X$ is executable' means that the command represented by $X$ is realized in some system execution.

To sum up, we postulate that our automata all satisfy:

1. Each node has a unique register value.
2. There are no redundant writes.
3. Unless otherwise stated all nodes are are assumed to be executable.

## 2.2 The accessibility relation

We are going now to discuss finite automata and point to an important property of them. The generalization of this property to infinite automata will be given subsequently. We start with a definition that makes sense in any automaton, finite or infinite.

**Definition 2.8** *Let $X, Y$ be nodes of an automaton. Say that 'Y is accessible from X', or $X \leq_{acc} Y$, iff there is a path from $X$ to $Y$, i.e., a path $x_0 = X, \ldots, x_n = Y$.*

The equivalence relation '$X$ is accessible from $Y$ as well as $Y$ is from $X$' (denoted $\equiv_{acc}$) splits the nodes into equivalence classes and these classes are partially ordered by: $\mathcal{X} \leq \mathcal{Y}$ iff for some $X \in \mathcal{X}$ and some $Y \in \mathcal{Y}, X \leq_{acc} Y$. If the automaton is finite, there is a maximal equivalence class—one with none above it—call it $\mathcal{E}$. Whenever $X$ is in $\mathcal{E}$ and $X \leq_{acc} Y$ then $Y \in \mathcal{E}$ too.

It may be useful to assume that each automaton forms a single equivalence class. In such a case we know that every two nodes are connected by a path. If the automata are finite then it is possible to get this situation as follows. (For simplicity we restrict our attention to a system of two automata $\mathcal{A}$ and $\mathcal{B}$.)

Let us assume that the total number of the nodes of the automata is minimal (for a solution to the critical-section problem). Let $\mathcal{E}$ be the maximal equivalence class of $\mathcal{A}$, and let $S$ be an external node in $\mathcal{E}$. $S$ must be executable, or else we would throw it away and find a smaller solution, in contradiction to the assumed minimality of the number of nodes. As we are going to show below, it is possible to make $S$ the start node of the automaton $\mathcal{A}$, and to find a new start node in the other automaton so that the two new automata still solve the critical-section problem. By minimality, $\mathcal{E}$ with the start node $S$, which forms the new automaton, cannot have fewer nodes then $\mathcal{A}$, and this implies that $\mathcal{A}$ itself consists of the single equivalence class, $\mathcal{E}$. Thus, for finite automata, we may assume
**the Accessibility property:** *All the nodes of the automata are executable and any two nodes in an automaton are accessible from one another.*

We now fill in some details on how to transform an external node into the start node of the automaton. Let $\mathcal{A}$ and $\mathcal{B}$ be automata that solve the critical-section problem, and $X$ be an external executable node in automaton $\mathcal{A}$. Let $\mathcal{A}'$ be the automaton resulting by deciding that $X$ with its register value is the start node of the automaton. We will find a new start node $Y$ in $\mathcal{B}$ such that the resulting automaton $\mathcal{B}'$ together with $\mathcal{A}'$ satisfy the mutual-exclusion and the lockout freedom properties. We argue as follows. Since $X$ is executable, there is in some system-execution an external event, $x$,that associated with the external edge leaving $X$. It is possible to change this system execution so that $\mathcal{A}$ remains forever in the external action $x$, or, at least, enough time so that $\mathcal{B}$ arrives to an external action $e$ (by the lockout freedom property). If $Y$ is the (external) node that $\alpha(e)$ leaves, and $\mathcal{B}'$ is the automaton obtained from $\mathcal{B}$ by making $Y$ the start node, it can be seen that $B'$ is as required.

In case the automaton is infinite, the Accessibility property may not be assumed and there is need for some substitute which we describe next. This substitute is, in our opinion, one of the main contributions of this paper, but the reader who wants a somewhat simplified proof may assume finite automata and the Accessibility property.

Let $P$ be a set of nodes in a given automaton $\mathcal{A}$. A node $S$ in $\mathcal{A}$ is said to *exclude* $P$ iff none of the nodes in $P$ is accessible from $S$. We say that $P$ is *dense* at $S$ iff no node accessible from $S$ exclude $P$. When $S$ is the start node, we say that '$P$ is dense (in $\mathcal{A}$)' instead of '$P$ is dense at $S$'

**Lemma 2.9** *Assuming, as we agrred, that all nodes in automaton $\mathcal{A}$ are executable. Let $\mathcal{P}$ be a finite list $\langle P_i \mid 1 \leq i \leq n\rangle$ of sets of nodes in automaton $\mathcal{A}$. Then there exists an external node $S$ in $\mathcal{A}$ with the following property: For any set $P_i \in \mathcal{P}$, either $S$ excludes $P_i$, or else $P_i$ is dense at $S$.*

The proof of the lemma is by a finite diagonalization. Define a sequence $X_i$ of nodes as follows. $X_0$ is the start node of $\mathcal{A}$. $X_{i+1}$ is defined as follows. If $P_{i+1}$ is dense at $X_i$, then $X_{i+1} = X_i$. Otherwise, let $X_{i+1}$ be a node accessible from $X_i$ that exclude $P_{i+1}$. Node $X_n$ has the desired property, and any external node accessible from it can be $S$ (since $X_n$ is executable, the lockout freedom implies that there is such an external node $S$).

## 2.3 The Cofinality relation

**Definition 2.10** *1. Given a system-execution we say that two event occurrences, $a$ and $b$, are* concurrent *iff $\neg(a \longrightarrow b) \& \neg(b \longrightarrow a)$. In such a case, for any representation of the system execution, the intervals of $a$ and $b$ overlap. If in some finite system execution, $\mathcal{S}$, $a$ and $b$ are maximal (no $c$ in $\mathcal{S}$ with $a \longrightarrow c$ or $b \longrightarrow c$), then $a$ and $b$ are said to be* cofinal *in $\mathcal{S}$. (In such a case they are necessarily concurrent.)*

2. *Two edges $A$ and $B$ in different automata (in some system of automata) are said to be* cofinal *iff in some system-execution there are two cofinal events, $a$ and $b$, with $\alpha(a) = A,\ \alpha(b) = B$.*

3. *If $X$ and $Y$ are two nodes in different automata, then we may also say that 'X and Y are cofinal' when some edges leaving $X$ and $Y$ are cofinal.*

Observe that the fact that $a$ and $b$ are concurrent in $\mathcal{S}$ does not imply that they are cofinal in some initial segment (see the Remark following Definition 2.4).

The following simple theorems are the main ingredients of the proof because it is here that the regularity of the register is used. We first present the finite case and then the general case (for infinite automata).

**Theorem 2.11** *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata, and suppose that the register of $\mathcal{B}$, the only one which $\mathcal{A}$ is reading, is boolean and regular. Assume the Accessibility property for $\mathcal{A}$ (defined in the previous subsection). Then any write edge of $\mathcal{B}$ is cofinal with any edge in $\mathcal{A}$.*

*Proof.* Let $W$ be any write edge of $\mathcal{B}$. Given any edge $X$ in $\mathcal{A}$, we will show that $W$ is cofinal with $X$. Take a system-execution, $\mathcal{S}$, (and a representation of it) in which an event $w$ is associated with $W$ (recall our assumption in Section 2.1 that all nodes are executable). There is an initial segment $\mathcal{S}_1$ of $\mathcal{S}$ in which $w$ is maximal (see the Remark following Definition 2.4). Let $t$ be the last event in $\mathcal{S}_1$ in $\mathcal{A}$, and set $T = \alpha(t)$. By the Accessibility property, $T \leq_{acc} X$, and there is in $\mathcal{A}$ a path leading from $T$ to $X$. By stretching the interval of $w$ beyond $t$ and letting $\mathcal{A}$ read from within that interval all the needed values (use regularity of the boolean register), it is possible for $\mathcal{A}$ to execute this path and arrive to a cofinal execution of the edge leaving $X$ with $w$. ∎

In the infinite case, the Accessibility property may no longer be assumed and we have the following substitute.

**Theorem 2.12** *Assume that $\mathcal{A}$ and $\mathcal{B}$ are two automata, and the register of $\mathcal{B}$, the only one which $\mathcal{A}$ is reading, is boolean and regular. Let $P$ be a dense set of nodes in automaton $\mathcal{A}$. Then any write node of $\mathcal{B}$ is cofinal with some node in $P$.*

*Proof.* Let $W$ be any (executable) write node of $\mathcal{B}$. We have to find some node $X$ in $P$ that is cofinal with $W$. Pick some system-execution, $\mathcal{S}$, (and a representation of it) in which an event $w$ is associated with the write edge of $W$. Let $\mathcal{S}_1$, $t$ and $T$ be as before. Since $P$ is dense, there is some $X \in P$ that is accessible from $T$. Now we let $\mathcal{A}$ read *from inside the interval of $w$* the values which are necessary for $\mathcal{A}$ to get to $X$, and thus make $X$ cofinal with $W$. (We use here the fact that the register of $\mathcal{B}$ is boolean and regular.) ∎

Even if $\mathcal{A}$ is an infinite automaton, $\mathcal{B}$ has a write edge that is cofinal with every edge of $\mathcal{A}$. This is the content of the following lemma.

**Lemma 2.13** *Let $\mathcal{A}$ and $\mathcal{B}$ be two automata where $\mathcal{B}$ has a boolean regular register that $\mathcal{A}$ is reading. Then any write node of $\mathcal{B}$ that is cofinal with the start node of $\mathcal{A}$ is cofinal with every node of $\mathcal{A}$.*

**Proof**. Observe first that a write node of $\mathcal{B}$ cofinal with the start node $S_A$ can be found by the following argument. Let $v_0$ be the register value of $S_A$ and let $P = P_B(S_B, v_0)$ be the path of $B$ obtained when $B$ only reads the initial value of $\mathcal{A}$. By the lockout freedom property, $P$ contains a critical section node $CS$. Now the register value of $CS$ is necessarily distinct from the initial value of $\mathcal{B}$ or else a contradiction to the mutual exclusion can be obtained. Thus the path $P$ contains a write edge which is as required. The proof of the Lemma itself now follow from the assumed regularity of the boolean register of $\mathcal{B}$.

Now we are going to explain how Lemma 2.9 and Lemma 2.13 will be used.

**Lemma 2.14** *Assume two automata $\mathcal{A}$ and $\mathcal{B}$ where the register of $\mathcal{B}$, which $\mathcal{A}$ is reading, is boolean and regular. Suppose a finite collection $\mathcal{P}$ of subsets of $\mathcal{A}$ and (with Lemma 2.9) let $S$ be an external node of $\mathcal{A}$ such that every set in $\mathcal{P}$ is either excluded by $S$ or else is dense at $S$. Let $\mathcal{A}'$ be the automaton obtained by taking only the nodes of $\mathcal{A}$ that are accessible from $\mathcal{S}$. Then there is an automaton $\mathcal{B}'$ (included in $\mathcal{B}$) such that the pair $\mathcal{A}'$, $\mathcal{B}'$ is a system that solves the critical section problem.*

*Proof.* The advantage of the new automata $\mathcal{A}'$ and $\mathcal{B}'$ is that now, for any $P \in \mathcal{P}$, if its intersection with $\mathcal{A}'$ is nonempty, then $P$ is dense in $\mathcal{A}'$.

By the lockout freedom, it is possible to find some external node in $B$ that is cofinal with $S$. Take any such external node $E$ in $\mathcal{B}$, and let $\mathcal{B}''$ be the new automaton obtained with this new external node as start node. The nodes of $\mathcal{B}''$ are all the nodes of $\mathcal{B}$ accessible from $E$.

$\mathcal{A}'$ and $\mathcal{B}''$ almost work, but there is a minor problem here: We found it convenient to assume that all the nodes of our automata are executable (Section 2.1), but it is possible that not all nodes of the new automata are executable, because, after choosing new start nodes, there are less system executions.

To solve this problem, we look at the pair $\mathcal{A}'$, $\mathcal{B}''$ of automata and throw away from $\mathcal{B}''$ all the nodes (and edges) that are now not executable. Let $\mathcal{B}'$ be the resulting automaton. To show that this suffices, we must prove that every node of $\mathcal{A}'$ is now executable. Since the start node of $\mathcal{A}'$ is certainly cofinal with some write node $W$ of $\mathcal{B}''$, the regularity of the boolean register implies (by Lemma 2.13) that every edge of $\mathcal{A}'$ which is accessible from the start node is cofinal with $W$, and is hence executable. Observe that since only $\mathcal{B}''$ was changed and $\mathcal{A}'$ remains, it is still true that every set in $\mathcal{P}$ is either excluded by $S$ or else is dense at $S$.

# 3 The boolean case

In this section we examine the situation of the critical section problem for two processes each possessing a boolean register. The answers given depend on the nature of the registers and on the assumptions made about the activity of the processes: are they allowed to stay forever in their external states or not.

It is easy to find a protocol for two processes that are active forever and have atomic registers. Below we will give a protocol for the somewhat harder case in which only one process is active forever and the other may go to sleep (the registers are boolean and atomic). However, if one of the registers is regular, then no solution is possible, even when both processes are assumed to be active forever. This is the main result of this section, and even though it is much simpler than our main impossibility result (the non existence of a protocol for regular registers of total size 5) it is not completely trivial, and it illustrates some of the techniques used for the harder case given in Section 4. Observe that the boolean case cannot be deduced from our main result of 4 which only applies to processes that may sleep forever in an external state—an assumption not needed here. It is not difficult to see that there is no solution to the critical section problem for two processes with boolean registers, even if both registers are atomic, if each of the processes may have a nonterminating external event.

## 3.1 A protocol for two processes with boolean atomic registers

The protocol is given in figure 0. Process $A$ is active forever (all of its actions, including the external stages, are terminating, and thus it repeatedly access the critical-section). The initial value of the register of $A$ is 1. Process $B$ may either not enter its critical section at all, or do so only finitely many times and then stay forever in an external state. The initial value of the register of $B$ is 0. As the lockout freedom property for each of the processes is easy to establish, let us only check the mutual exclusion property.

*figure 0*

Let us introduce some notation related to a system execution $\mathcal{S}$ that executes the protocol. Denote with $r_1$, $w_0$, $r_0$, and $w_1$ the events corresponding to the read edge $\rho_1$, the write edge $\omega_0$, the read edge $\rho_0$, and the write edge $\omega_1$ respectively (in $A$). The $i$-th execution ($i \geq 1$) of the read $\rho_1$ in $\mathcal{S}$ is denoted $r_1[i]$; the meaning of $w_0[i]$, $r_0[i]$, and $w_1[i]$ is similarly defined. In a similar vein define $t_1[i]$, $s_0[i]$, $t_0[i]$, and $s_1[i]$ to be the $i$-th events of $B$ corresponding to $\tau_1$, $\sigma_0$, $\tau_0$ and $\sigma_1$ respectively.

**Lemma 3.1** *For every $i \geq 1$*

$$t_1[i] \longrightarrow r_1[i] \longrightarrow w_0[i] \longrightarrow s_0[i] \longrightarrow t_0[i] \longrightarrow r_0[i] \longrightarrow w_1[i] \longrightarrow s_1[i]$$

The proof can be carried out by induction on $i$. Since the critical-section events of $B$ are always situated in between $s_0[i]$ and $t_0[i]$, and as the interval between $w_0[i]$ and $r_0[i]$ includes the former interval but contains no critical-section events of $A$, the mutual-exclusion property follows.

## 3.2 Non existence of boolean solutions when one register is regular

**Theorem 3.2** *There is no critical section protocol for two processes with boolean registers that satisfies the mutual exclusion and the lockout freedom property if one of the registers is regular. This holds even if the processes are assumed to be active for ever.*

To isolate those ingredients that are peculiar to the analysis of infinite automata, we first prove this theorem for finite automata, and then indicate the changes made to accommodate the infinite case. But first we are going to make a general observation which relies not on the size or the assumed regularity of the registers, and which holds for any two automata (finite of infinite) that solve the critical section problem.

Let us assume two protocol automata, I and II, that solve the critical section problem. Recall our assumption that all nodes are executable, that each node has some register value, and that no protocol writes unless it changes the value of its register.

**Lemma 3.3** *Let $X$ be either of protocol automata I or II.*

1. *For any node $N \in X$, there is a value $v$ such that $P_X(N, v)$ contains a write node (other than $N$, if $N$ is a write node).*

2. *There is a write node $T$ and a value $v$ such that $LP_X(T, v)$ contains a critical-section node $C$ (there is no write node (strictly) in between $T$ and $C$—see Definition 2.2).*

*Proof.*

It is quite obvious that from any node there is a path leading to a write node (a path which may consists of several reads of different values); the point of this Lemma is to show that a single value suffices. It is also clear that there is a write node that leads, either directly or through a a path of only read edges, into a critical-section node; part *2* of this Lemma shows that, at least for some write node, a single value $v$ may suffice.

Observe first that is suffices to prove *1* under the assumption that $N$ is not a write node (for otherwise look at the node following $N$—if it is a write node, we are done, and otherwise apply *1* to it). Let $N$ be a node in automaton $X$, and assume that for no value $v$ does $P_X(N, v)$ contain a write node. (And $N$ is not a write node.) We will derive a contradiction by constructing a system-execution in which the mutual-exclusion does not hold. Let $Y$ be the other automaton.

Since $N$ is executable, there is a finite system-execution $\mathcal{S}_0$, so that the last event of $X$ in $\mathcal{S}_0$ is associated to $N$. Let $t$ be the last event of $Y$ in $\mathcal{S}_0$. Let $T$ be the edge in $Y$ corresponding to $t$. Let $v_N$ be the register value of $N$. Look at $P = P_Y(T, v_N)$. Since the range of values of the register of $Y$ is finite, for some value $s$ this infinite path in $Y$ must contain infinitely many nodes with register value $s$. By our assumption, $Q = P_X(N, s)$ contains no write nodes, and hence all of its nodes remain with register value $v_N$. Now an infinite system-execution $\mathcal{S}$ can be defined that extend $\mathcal{S}_0$ and in which, after $\mathcal{S}_0$, the register value of $X$ is never changed.

By the lockout freedom, the paths $Q$ and $P$ (executed in $\mathcal{S}$) contain infinitely many critical-section nodes. The contradiction is obtained by describing a system execution in which a pair of these critical section nodes is concurrent. So let $\mathcal{S}_1$ be an initial segment of $\mathcal{S}$ in which the last event of $X$ is an execution of a critical section edge (with register value $v_N$). Now just extend $\mathcal{S}_1$ by fixing $X$ and allowing $Y$ to follow the path $P$ by reading $v_N$, until it arrives to xome critical section node. We shall prove now *2* by deriving a contradiction from the assumption that, for every write node $T$ in $X$ and value $v$, there is no critical-section node in $LP_X(T, v)$. We will define a system-execution $\mathcal{S}$ in which $X$ is locked-out. $\mathcal{S}$ is defined by pasting together (concatenating) a sequence, $\mathcal{S}_i$, of system-executions defined by induction. The path followed by $X$ in each $\mathcal{S}_i$ is of type $LP_X(T_i, v_i)$, and our assumption thus imply that $X$ is locked out.

$\mathcal{S}_i$ contains,

(1) in $X$: a write-of-$t_i$ event, denoted $e(T_i)$ and associated to edge $T_i$, and a group of read events $\overline{r_i}$, which are all read-of-$v_i$; and

(2) in $Y$: a sequence $\overline{y_i}$ of reads-of-$t_i$ followed (unless $v_i$ is already the value of $Y$'s register) by a write-of-$v_i$ event, $e(S_i)$, corresponding to the edge $S_i$. The temporal order of these events is: $e(T_i) \longrightarrow \overline{y_i} \longrightarrow e(S_i) \longrightarrow \overline{r_i}$.

Well, $T_0$ is the initializing write edge of $X$ and $t_0$ is the initial value which is written by $T_0$. Let $v_0$ be a register value appearing infinitely often in $P_Y(S, t_0)$, where $S$ is the initializing write edge of $Y$. We claim that $P_X(T_0, v_0)$ contains a write edge $T_1$ other than $T_0$. If not, it is possible to describe a system execution in which $X$ follows $P_X(T_0, v_0)$, and $Y$ follows $P_Y(S, t_0)$. But then $X$ is locked out, and this contradiction proves our claim. Now let $S_0$ be the first write-of-$v_0$ in $P_Y(S, t_0)$. The definition of stage $i+1$ is similarly done, and a contradiction is thus derived because a system execution is obtained in which $X$ is locked out. ∎

We now return to the proof of Theorem 3.2. Assume two automata $I$ and $II$, with boolean registers, and with a regular register for $I$.

**THE FINITE CASE**. In the finite case, the Accessibility property can be assumed (see Definition 2.2; in fact only the accessibility for $II$ is needed).

First we define the values 0 and 1. By Lemma 3.3(*2*), let $T_1$ be a write edge in I so that, for some value $x_1$, $LP_I(T_1, x_1)$ contains a critical section node. Similarly, let $x_2$ and $T_2$ be such that $LP_{II}(T_2, x_2)$ contains a critical section node. Then call the value written by $T_1$, $1_I$, and the value written by $T_2$, $1_{II}$. In fact, we will omit the subscript, and simply write 1. Then the other value is denoted 0.

**Lemma 3.4** *Let $X$ be either of the protocols I or II.*

1. *For every write edge $W$, $P_X(W, 1)$ contains no critical-section node.*

2. *For every write-of-0 edge $W$ in $X$ and value $v$, $LP_X(W, v)$ contain no critical-section node.*

3. $x_1 = 0$ *and* $x_2 = 0$.

*Proof.* By its definition, $LP_{II}(T_2, x_2)$ contains a critical-section node, $CS_{II}$, and its register value is 1. (by definition of the limited path, $T_2$ is the only write edge in that path).

As the register of I is boolean and regular, and as automaton II is forming a single equivalence class (in $\equiv_{acc}$), any write edge $W$ in I is cofinal with any edge in II, and in particular with the edge of $CS_{II}$. Hence, by the mutual-exclusion property, $P_I(W, 1)$ contains no critical-section node. This proves *1* for process I.

Recall that $T_1$ is a write-of-1 edge in I such that for some value $x_1$, $LP_I(T_1, x_1)$ contains a critical section node. By what we have just proved above, $P_I(T_1, 1)$ contains no critical-section node, and hence $x_1 = 0$. So

$$LP_I(T_1, 0) \text{contains a critical-section node.}$$

This proves the first equality in *3*.

Now we prove *2* for II by contradiction. Assume that $LP_{II}(W, v)$ contains a critical-section node, $C_0$, where $W$ is a write-of-0 edge in II. $C_0$ has register value 0 But $T_1$, as any write of I, is cofinal with the critical-section node $C_0$, and this contradicts what we already know, that $P_I(T_1, 0)$ contains a critical-section node.

To prove *1* for II, assume towards a contradiction that $P_{II}(W, 1)$ contains a critical-section node, for some write edge $W$ in II. By taking $W$ to be the last write edge in that path (before the critical section), we may assume that $LP_{II}(W, 1)$ contains a critical-section node. It follows from *2* that $W$ is a write-of-1 edge. Let $N$ be the node that the edge $W$ is leaving. By our assumption that any write changes the register's value, the register value of $N$ is 0. Let $K$ be an edge abutting on $N$. Since the write node $T_1$ is cofinal with $K$, a contradiction to the mutual-exclusion property can be obtained (since both $LP_I(T_1, 0)$ and $P_{II}(W, 1)$ contain a critical-section node).

Now, to prove *3* for II, recall that $T_2$ is a write edge in II such that for some value $x_2$ $LP_{II}(T_2, x_2)$ contains a critical section node. We claim that $x_2 = 0$. If not, then $x_2 = 1$ is in contradiction to *1* for $II$.

It remains to show that *2* holds for $I$. If for some write-of-0, $W$, $LP_I(W, v)$ contains a critical-section node, then by *1*, $v = 0$. By definition of the limited path, this critical-section node is of register value 0. Let $N$ be the node of $II$ that $T_2$ is leaving, then the register value of $N$ is 0. By regularity of the register of $I$, there is a system execution in which the last event by $I$ is an execution of $W$, and $II$ is at the state corresponding to $N$. Now $I$ continues from $W$ and reads 0's until it gets to the critical-section node in $LP_I(W, 0)$. This critical-section node has register value 0. Then $II$ continues from $N$, writes the 1 in executing $T_2$ and reads the 0's until it gets too into the critical-section node. ∎

It is possible at this stage to give an intuitive overview of the proof of Theorem 3.2. We saw that both of the protocol automata contain write-of-1 such that reads of 0 lead to the critical-sections. Now what happens in case both protocols show their 1's to each other? Who is to yield? We are going to show that the one who yields is locked-out.

**Definition 3.5** *Let $W$ be a write node (or edge) in automaton $X$. We say that $X$* yields *at $W$ iff $P_X(W, 1)$ stabilizes at 0, i.e., contains only finitely many nodes with register value 1.*

**Lemma 3.6** *Either I yields at every write node, or II yields at every write node.*

*Proof.* Suppose, for some write node $W_0$ in $I$, $I$ does not yield at $W_0$. So by definition, $P = P_I(W_0, 1)$ contains infinitely many nodes of register value 1, and hence at least one write-of-1 node which we may assume to be $W_0$ itself.

Let $W_i, i = 1, \ldots,$ denote these infinitely many nodes in $P$ with register value 1.

We will show that II yields at each of its write nodes. Assume that it does not, and let $T_0$ be some write of II such that II does not yield at $T_0$. So $P_{II}(T_0, 1)$ contains infinitely many nodes with register value 1. As any node in II, $T_0$ is cofinal with $W_0$.

We can define now a system-execution $\mathcal{S}$ in which I executes the path $P_I(W_0, 1)$, which contains no critical-section node (by 3.4(1)). In $\mathcal{S}$, $II$ reads 1's from the $W_i$'s, and supplies the needed values of 1 by executing $P_{II}(T_0, 1)$. In this system-execution I is locked-out. ∎

The proof of the Theorem is concluded by the following Lemma.

**Lemma 3.7** *If automaton $X$ yields at each write node, then $X$ is locked-out in some system-execution.*

*Proof.* We will define an executable path, $A$, in automaton $X$ which includes no critical-section nodes. $A$ includes a sequence $W_i$ of write-of-1 edges defined inductively. (See figure 1.) $W_0$ is any write-of-1 in $X$. Suppose that $W_i$ is defined. We know that $X$ yields at $W_i$; define $Z_i$ to be the stable write-of-0 in $P = P_X(W_i, 1)$. That is, $Z_i$ is the last write in $P$ and it is a write-of-0 edge. By Lemma 3.4(1), $P$ contains no critical-section node. Let us denote with $r_i$ the group of reads-of-1 that moved the automaton from $W_i$ to $Z_i$.

By Lemma 3.3(1), for some value, $v$, $P_X(Z_i, v)$ contains a write edge after $Z_i$, and the first one has to be a write-of-1, since it changes the register's value. Since $Z_i$ is a stable write-of-0 in $P$, $v = 0$. So let now $W_{i+1}$ be the first write-of-1 in $Q = P_X(Z_i, 0)$. By Lemma 3.4(2), we know that $Q$ contains no critical-section node in between $Z_i$ and $W_{i+1}$. Let us denote with $s_i$ the group of reads-of-0 edges that caused the change from $Z_i$ to $W_{i+1}$. The path $A$ is the path in automaton $X$ that includes all the $W_i$ defined above. If we are able to define a system-execution $\mathcal{S}$ in which $X$ follows the path $A$, then $X$ is locked-out in $\mathcal{S}$.

To define $\mathcal{S}$, let us define a path, $B$, in the other automaton $Y$, that supplies the values needed by the reads in $A$. Inductively, define write edges in $Y$, $R_i$ and $S_i$. $R_i$ is a write-of-1 and supplies the values needed by the read(s) $r_i$, and $S_i$ is a write-of-0 needed by the read(s) in $s_i$. $R_0$ is any write-of-1 that follows $W_0$ in some system execution (the next argument also shows how to get $R_0$). Suppose that $R_i$ is defined and $\overline{W}_i \longrightarrow \overline{R}_i \longrightarrow \overline{r}_i$ in the system-execution so far constructed (where the notation $\overline{U}$ denotes the event executing the edge $U$). By Lemma 3.3(1), there is a value $v = 0, 1$ such that $P_Y(R_i, v)$ contains a write-of-0 node. It is possible to place the read(s) of $v$ either before the execution of $Z_i$, in case $v = 1$, or after $Z_i$ in case $v = 0$. $S_i$ is the write-of-0 thus obtained. Set $\overline{S}_i \longrightarrow \overline{s}_i$. Then $R_{i+1}$ is defined as a write-of-1 following $S_i$ and $\overline{W}_{i+1} \longrightarrow \overline{R}_{i+1}$. The existence of $R_{i+1}$ is derived by using again Lemma 3.3(1). ∎

*figure 1*

**THE INFINITE CASE** The structure of the proof of Theorem 3.2 for the infinite case is the same as for the finite case, but the Accessibility property may no longer be assumed. The idea of the proof in the infinite case is to observe that, whenever the Accessibility property was used, it was only required that a certain set is dense in II. So assume two protocol automata, I and II, with boolean registers and with I having a regular register. The register of II may be atomic. We may assume that all edges are executable, that each node has some register value, and that no protocol writes onto its register unless it changes the value of that register.

We are going to use Lemma 2.9 for the following list $\mathcal{P}$ of subsets of II.

1. For any pair of register values $x$, and $y$, let $P_{x,y} = \{T \mid T$ *is a write-of-*$x$ *node in* II *such that* $LP_{II}(T, y)$ *contains a critical-section node*$\}$.
2. Also define $B_{x,y}$ to be the set of write nodes $T$ in II such that $P_{II}(T, x)$ contains infinitely many nodes of register value $y$.

Lemmas 2.9 and 2.14 give two new automata obtained by choosing cofinal external nodes as the new start nodes, so that now, whenever $P_{x,y}$ or $B_{x,y}$ is non-empty, then it is dense. (We continue to denote by I and II these new automata.)

The nodes $T_1$ and $T_2$ and the values 0 and 1 are defined again so that $T_1$ and $T_2$ are write of 1, and $LP_I(T_1, x_1)$, and $LP_{II}(T_2, x_2)$ contain critical section nodes. $T_2 \in P_{1,x_2}$, and thus, by our special assumption (the use of Lemma 2.14), $P_{1,x_2}$ is dense in II.

Lemma 3.4 continues to hold in the infinite case as well, but some changes must be made in its proof. For completeness, we reprove the Lemma.

*Proof of Lemma 3.4 for the infinite case.* As $T_2 \in P_{1,x_2}$, $P_{1,x_2}$ is dense and so the set of critical-section nodes in II with register value 1 is dense too. Thus any write node $W$ in I is cofinal with such a critical-section node (by Theorem 2.12, as the register of I is regular), and hence (by the mutual-exclusion property) $P_I(W, 1)$ contains no critical-section node. This proves *1* for process I. In particular, $P_I(T_1, 1)$ contains no critical-section node, and hence $x_1 = 0$. So $LP_I(T_1, 0)$ contains a critical-section node. This proves *3* for I.

Now we prove *2* for II. For the sake of a contradiction assume that $LP_{II}(V, v)$ contains a critical-section node where $V$ is a write-of-0 edge in II. Thus $P_{0,v}$ is non-empty and is hence dense. So the set of critical-section nodes with register value 0 is dense, and so (by Theorem 2.12) the write node $T_1$ of $I$ is cofinal with some such critical-section node. But this contradicts what we already know, that $P_I(T_1, 0)$ contains a critical-section node.

To prove *1* for II, assume towards a contradiction that $P_{II}(W,1)$ contains a critical-section node, for some write node $W$. By taking $W$ to be the last write node in that path, we may assume that $LP_{II}(W,1)$ contains a critical-section node. It follows from *2* that $W$ is a write-of-1 edge, and hence $P_{1,1}$ forms a dense set. Let $N$ be a node in $P_{1,1}$ that is cofinal with $T_1$. As the write edge of $N$ is a write-of-1, the register value of $N$ is 0 (by our assumption that any write changes the value of the register). A contradiction to the mutual-exclusion property can be obtained since both $LP_I(T_1,0)$ and $P_{II}(N,1)$ contain a critical-section node.

Now we prove that $x_2 = 0$. If not, then $x_2 = 1$ and $LP_{II}(T_2,1)$ contains a critical section node. This is in contradiction to *1* for II.

It remains to show that *2* holds for I. If for some write-of-0, $W$, $LP_I(W,v)$ contains a critical-section node, then by *1* $v = 0$. By definition of the limited path, this critical-section node is of register value 0. Since $x_2 = 0$, the set of nodes $P_{1,0}$ is dense in $II$. This gives a contradiction as follows. Let $N \in P_{1,0}$ be cofinal with $W$. Since $N$ is a write-of-1 node, the register value of $N$ is 0. It is possible to form a system execution in which I continues from $W$ and reads 0's until it gets to the critical-section node in $LP_I(W,0)$. This critical-section node has register value 0. Then II continues from $N$, write the 1 and reads the 0's till it gets too into the critical-section node. ∎

The proof of Lemma 3.6 for the infinite case is modified as follows. Instead of using the Accessibility property and the regularity of I's register to conclude that $T_0$ and $W_0$ are cofinal, the density of $B_{1,1}$ is deduced from the existence of $T_0$. Then, as before, by the regularity of I's register, $W_0$ is cofinal with *some* $T_0 \in B_{1,1}$ (Theorem 2.12). Now the proof of 3.6 continues as before. There are no changes in the proofs of Lemma 3.7, and thus Theorem 3.2 is proved. ∎

# 4 The 2-3 impossibility proof for regular registers

**Theorem 4.1** *1. There is no protocol for the critical-section problem for n processes that use regular registers, if the total size of the registers is* $< 3n$.

*2. There is no protocol for the critical-section problem for two processes that use registers of size two and three if the 2-valued register is regular. (Even if the 3-valued register is atomic).*

As we already argued in Lemma 2.5, it is enough to prove *2*. Assume for sake of a contradiction the existence of a protocol consisting of two automata that satisfy the lockout-freedom and the mutual-exclusion property, and that use only two and three values in their registers. Denote by II that automaton with a two-valued regular register and by III the one with a three-valued atomic register. The nodes of II are written in sanserif letters, and those of III in italics (but we nowhere rely on this assumption). Recall that we may assume that each node in the automata corresponds to a state and thus has a unique register value. The register value of the start node is the initial value, and any write edge going from node $A$ to $B$ changes the register value of node $B$ according to its tag value. An assumption continuously used in the proof is that any of the processes may stay forever in an external state.

The automata are not assumed to be finite, and Lemma 2.9 is used for a finite list of sets of nodes in III. Instead of giving the tedious list now, we just assume a list $\mathcal{P}$ with all the needed sets; and defer the definition of this list to the end of the proof. The reader should have no difficulties in verifying that each set used during the proof indeed appears in Definition 4.15.

Using Lemma 2.14, new start nodes are assigned, $\mathsf{S}$ to II and $S$ to III, so that the mutual-exclusion and the lockout freedom property hold, and so that any $P \in \mathcal{P}$ is either excluded at $S$, or else is dense at $S$. We shall say that $P$ is 'dense' if it is dense at $S$. Thus, whenever there is some $X \in P$ that is accessible from $S$, then $P$ is dense and so for any node $C$ accessible from $S$ there exists some $D$ in $P$ accessible from $C$.

Let us define the values 0, 1 and 2 for the two automata. Denote by 0 the initialization value of II's register; that is, the register value at the start node $\mathsf{S}$. Then 1 is simply the other value of II. Now for III define the values 0,1 and 2 as follows. By the lockout freedom assumption, $P_{III}(S, 0)$ contains infinitely many critical-section nodes and external nodes between them. (To see this, define a system-execution in which II contains only one event—the external event associated with $\mathsf{S}$—and such that all the read events of III only return 0. Thus all the events of III in this system-execution are in $P_{III}(S, 0)$, and by the lockout-freedom property the conclusion follows.) For two values, denoted 0 and 1, the path $P_{III}(S, 0)$ contains infinitely many critical-section nodes of register value 1, and external nodes of register value 0. It is clear that $0 \neq 1$ since otherwise a violation of the mutual-exclusion property can be obtained. We call the third value of III's register: 2. By picking any one of these external nodes of III as the start node, we may assume that the register value of $S$ itself is 0.

We already argued (Lemma 2.14, but it does no harm to repeat) why all the nodes are executable: For II, we did throw away all inexecutable nodes (and redirect all inexecutable read edges backwards to their read nodes). For III, we even have the following.

**Proposition 4.2** *Let* $\mathsf{W}$ *be the first write node in* $P = P_{II}(\mathsf{S}, 0)$ *(the first write node is taken for definiteness—any other write node in* $P$ *works); then* $\mathsf{W}$ *is cofinal with every node in III.*

Since $P$ is executed in a system execution in which III remains in its external state determined by $S$, $P$ necessarily contains a write node. The proof is an easy consequence of the assumed regularity of the boolean register of II. The Proposition implies that all the nodes of III are executable. We fix from now on $\mathsf{W}$ to be the first write node in the path $P$. $\mathsf{W}$ is a write-of-1 since it changes the register's value.

Before continuing, an intuitive overview of the proof is in place. When II reads the value 0, it may believe that III is in an external state of value 0 and advance to a critical-section state; when II reads the value 1, it is prevented from entering a critical-section state since III may be in one of the critical-section states of value 1. But what happens if II reads the value 2? We shall prove that II contains a write node $\mathsf{V}$ such that $P_{II}(\mathsf{V}, 2)$ contains a critical-section node (Proposition 4.10). Turning to III, we ask what can be the register values with which it enters a critical-section state? We don't expect 0 to be such a value as II is prompted by 0 to enter a critical-section state. It may be 1 by definition, but could it be 2? We shall prove (and this is our main Lemma 4.14) that III may enter a critical-section state with register value 2. Not only that, but we may assume that the write $T$ of value 2 just preceding this critical-section state, and the write $\mathsf{V}$ of II mentioned above, are cofinal. Now the contradiction is derived as follows: We may assume that the interval of $\mathsf{V}$ is somewhat longer, and III may read from within this interval any sequence of values needed to advance from $T$ into a critical-section state, and then II reads the value 2 needed to enter its critical-section state, and thus the mutual-exclusion property is violated. The details follow.

**Lemma 4.3** *If $E$ is a node in III with register value 0, then $P_{III}(E, 1)$ contains no critical-section node.*

*Proof.* We claim first that there are infinitely many critical-section nodes in $P = P_{II}(\mathsf{S}, 0)$ and they all have register value 1. Since $P$ can be obtained as the path resulting in a system-execution that fixes III to its start position $S$ (which has register value 0), it is clear that $P$ must contain unboundedly many critical-section nodes. The register value of any such a critical-section node cannot be 0, since otherwise by activating III from $S$ and presenting this 0 to the reads of III, it too will reach a critical-section node, contradicting the mutual-exclusion property. Hence the register value of any critical-section in $P$ is 1. Recall that $\mathsf{W}$ is the first write-of-1 node in $P$.

Assume in contradiction to our Lemma that the path $P_{III}(E, 1)$ reaches a critical-section node. By Proposition 4.2, there is a finite system-execution in which II's last event is an execution of the write-of-1 $\mathsf{W}$, and III has reached $E$ and stopped. Now let II continue and read the 0 of $E$ till it gets into a critical-section node of register value 1 (as shown above). Then let III continue, read 1's, and enter its critical-section node too in contradiction to the mutual-exclusion property. ∎

**Definition 4.4** *1. A node $ST$ of III is called a $START$ node iff $ST$ is external, has register value 0, and the path $P_{III}(ST, 0)$ contains infinitely many external nodes of register value 0 and critical-section nodes of register value 1. (The unique edge leaving a $START$ node is called a $START$ edge.)*

*2. A node of II is called $START$-cofinal iff it is cofinal with some $START$ node.*

Obviously, $S$ (the start node of III) is a $START$ node. As property $START$ is in $\mathcal{P}$ (defined in item 1 of 4.15), $START$ is a dense property, and a $START$ node is accessible from any node of III. Theorem 2.12 implies that any write node of II is cofinal with some $START$ node. That is, any write node of II is $START$-cofinal . Let us record this in a Lemma.

**Lemma 4.5** *Any write node of II is $START$-cofinal. The start node $\mathsf{S}$ of II is also $START$-cofinal (as it is cofinal with $S$).*

**Proposition 4.6** *Any $START$-cofinal external node of II has register value 0. Any $START$-cofinal critical-section node has register value 1.*

*Proof.* Let $\mathsf{X}$ be a $START$-cofinal external node of II and $ST$ be a $START$ node of III cofinal with $\mathsf{X}$. Since $P_{III}(ST, 1)$ contains no critical-section nodes (by Lemma 4.3), and since II can stay forever in the external node $\mathsf{X}$, $\mathsf{X}$ cannot have register value 1.

Now let $\mathsf{C}$ be a $START$-cofinal critical-section node, and $ST$ a $START$ node of III cofinal with $\mathsf{C}$. Since $P_{III}(ST, 0)$ contains a critical-section node, $\mathsf{C}$ cannot be of register value 0, or else the mutual-exclusion property is violated in some easily defined system-execution. ∎

**Proposition 4.7** *1. If $\mathsf{N}$ is any $START$-cofinal node of II then $P_{II}(\mathsf{N}, 0)$ contains a critical-section node and an external node afterwards; and both nodes are $START$-cofinal. (By the previous proposition, they have register values 1 and 0 respectively.)*

*2. If $\mathsf{T}$ is a write node of II or is a $START$-cofinal external node, then $P_{II}(\mathsf{T}, 1)$ contains no critical-section nodes and stabilizes at value 1 (see Definition 2.7 for* stabilization*).*

*Proof.* Given $\mathsf{N}$ as in *1*, let $ST$ be some $START$ node of III cofinal with $\mathsf{N}$. Since III can stay forever in the external node $ST$, and since any $START$ node has register value 0, the first proposition is a consequence of the lockout-freedom property. It is clear that all nodes in $P_{II}(\mathsf{N}, 0)$ are $START$-cofinal too.

Let $\mathsf{T}$ be a write node of II. To see that $P_{II}(\mathsf{T}, 1)$ contains no critical-section node, it suffices to find a critical-section node $E$ in III cofinal with $\mathsf{T}$ and with register value 1. As any write node of II, $\mathsf{T}$ is cofinal with nodes in every dense set of $\mathcal{P}$ (Theorem 2.12). The second set in Definition 4.15, for $w = 1$ and $v = 0$, is the set of all critical-section nodes $E$ in III with register value 1 and such that $P_{III}(E, 0)$ contains infinitely many critical-section nodes of value 1. This set is dense since by the definition of 0 and 1, any critical-section node of register value 1 in $P_{III}(S, 0)$ is in this set. Hence we conclude that $\mathsf{T}$ is cofinal with some critical-section node with register value 1, $CS_1$, of III such that $P_{III}(CS_1, 0)$ contains infinitely many critical-section nodes with register value 1. (In case $\mathsf{T}$ is a $START$-cofinal external node, then it has register value 0 (by Proposition 4.6) and the same conclusion can be derived by definition of $START$.) The stabilization at 1 follows by assuming that $P_{II}(\mathsf{T}, 1)$ contains infinitely many nodes of register value 0. We will see in such a case that II can be locked-out.

As $\mathsf{T}$ is cofinal with $CS_1$, there is a finite system-execution $\mathcal{S}$ such that the last events in II and III there are $t$ and $c$, the cofinal events such that $\mathsf{T} = \alpha(t), CS_1 = \alpha(c)$. Continue the system-execution by alternatingly activating II and III, so that II follows the path $P_{II}(\mathsf{T}, 1)$, and III follows the path $P_{III}(CS_1, 0)$. To do this, let II read the register value 1 of $CS_1$ until it arrives (by our assumption) to the first node of register value 0. Then let III read 0 until it gets to the next critical-section node of register value 1. Then activate II again, and so on. Since II only reads 1 in this execution of the path $P_{II}(\mathsf{T}, 1)$, it never reaches a critical-section node as we argued above, and this contradicts the lockout-freedom property. ∎

**Proposition 4.8** *If $E$ is any node of III, then $P_{III}(E, 1)$ stabilizes at 2.*

*Proof.* Let $E$ be a node of III. To get a contradiction, assume that $P_{III}(E, 1)$ contains infinitely many nodes with register value 0 (this is case 0) or infinitely many nodes of register value 1 (this is case 1).

Recall that $\mathsf{W}$ is a write-of-1 node of II which is cofinal with any node of III and in particular with $E$ (Proposition 4.2). Then $\mathsf{W}$, as any write, is $START$-cofinal. Hence, $P_{II}(\mathsf{W}, 0)$ contains infinitely many critical-section nodes of value 1 (Propositions 4.7(1)), and $P_{II}(\mathsf{W}, 1)$ stabilizes at 1 (Proposition 4.7(2)). Assume a system-execution and a representation of it in which $\mathsf{W}$ is cofinal with $E$.

In case 0, let III read 1's (following the path $P_{III}(E, 1)$) till the first of the infinitely many 0's nodes. We can describe a system-execution in which II reads 0's (from $\mathsf{W}$ onwards) and enters infinitely many times critical-section nodes of register value 1, and III reads these 1's, produces zeros but (from some moment on) never gets into a critical-section node (by Lemma 4.3 applied to those value 0 nodes). Thus III is locked-out.

In case 1, let III read 1 until it gets to the first of the infinitely many nodes of value 1 in $P_{III}(E,1)$, and then activate II and continue with $P_{II}(\mathsf{W},1)$ until stabilization at 1. Now II is locked-out: III produces infinitely many nodes of register value 1 in $P_{III}(E,1)$, and II never changes the 1 in its stable register while reading those 1's, and contains no critical-section nodes (Proposition 4.7(2)). A contradiction which proves the proposition. ∎

In particular, let $ST$ be any $START$ node of III. Proposition 4.8 implies that $P = P_{III}(ST,1)$ stabilizes at 2 and, by Lemma 4.3, $P$ contains no critical-section node. Let us denote by $U = U(ST)$ the stable write-of-2 node in $P$. So all nodes in $P$ following $U$ have register value 2.

**Corollary 4.9** *If* $\mathsf{V}$ *is a write node by II, then* $P_{II}(\mathsf{V},2)$ *stabilizes at 0.*

*Proof.* As any write of II, $\mathsf{V}$ is $START$-cofinal. Let $ST$ be a $START$ node of III cofinal with $\mathsf{V}$. Put $U = U(ST)$. By definition, $U$ is a write-of-2 node in $P = P_{III}(ST,1)$ such that $P_{III}(U,1)$ contains no write of value $\neq 2$ (and no entry to critical-section, by Lemma 4.3, which is applicable as $P_{III}(U,1)$ is contained in $P$). If $P_{II}(\mathsf{V},2)$ contains infinitely many nodes of register value 1, a contradiction is derived by showing that III may be locked-out. ∎

**Proposition 4.10** *II contains a write-of-1 node* $\mathsf{V}$*, such that* $LP_{II}(\mathsf{V},2)$ *contains a critical-section node (with register value 1).*

*Proof.* We shall define an infinite path $P = P_{II}(\mathsf{S},\sigma)$in II by means of a sequence $\sigma$ of 1's and 2's that II is reading, and deduce that it contains a write node as required.

The path starts with $\mathsf{S}$, the start node of II, and reads 1's until the first write-of-1. Then it reads 2's until the first write-of-0; and so on alternating reads of 1's and 2's whenever a new write occurs. Formally, let us define a sequence of write nodes of II, $\mathsf{W}_1, ..., \mathsf{W}_i ...$ where $\mathsf{W}_i$ is a write-of-($i \ mod \ 2$) node, as follows. Let $\mathsf{W}_{i+1}$ be the first write after $\mathsf{W}_i$ in $P_{II}(\mathsf{W}_i, 1+a)$ where $a$ is the value written by $\mathsf{W}_i$. In case $\mathsf{W}_i$ is a write-of-0, $P_{II}(\mathsf{W}_i,1)$ contains a write-of-1 by Proposition 4.7(2). And in case $\mathsf{W}_i$ is a write-of-1, $P_{II}(\mathsf{W}_i,2)$ contains a write-of-0 by Proposition 4.9.

There is a system-execution which executes this path $P$ defined by the $\mathsf{W}_i$'s; that is, we can figure out how III supplies these values of 1's and 2's. Indeed, reading the value 0 of $\mathsf{S}$, III gets to a write-of-1 node in $P_{III}(S,0)$ ($S$ is the start node of III, and we remarked that, by its definition, it is a $START$ node and hence $P_{III}(S,0)$ contains a write-of-1 node). Then II reads this 1 till it gets to $\mathsf{W}_1$, the first write-of-1. Then III can read the value 1 of $\mathsf{W}_1$ until it gets to a node of value 2 (by Proposition 4.8), and then, during the write event connected to $\mathsf{W}_2$, III reads all the necessary values to arrive to a node of register value 1. (Use the fact that the set of write-of-1 nodes of III are in $\mathcal{P}$ and are dense.) In this manner, the path of III can be defined, and a system execution exists in which these paths are followed.

By the lockout-freedom property, $P$ contains infinitely many critical-section nodes and external nodes. By Proposition 4.7(2), critical-section nodes in $P$ cannot be found in $P_{II}(\mathsf{W}_i, 1)$ where $\mathsf{W}_i$ is a write of 0 or is $\mathsf{S}$, and thus a critical-section node exists in an interval of $P$ determined by $P_{II}(\mathsf{W}_i, 2)$ where $\mathsf{W}_i$ is a write-of-1 node. This interval of $P$ contains no write nodes, as $\mathsf{W}_{i+1}$ is by definition the first write there. Hence the critical-section node that we found has to have register value 1. ∎

**Definition 4.11** *A node $E$ of III is* normal *iff $P_{III}(E, 1)$ contains no critical-section node.*

**Corollary 4.12** *Any write node of III is normal.*

*proof.* We proved in Lemma 4.3 that any node $E$ of register value 0 is normal (and hence any $START$ node is normal). Let $T$ be a write node of III. It thus follows that if the register value of $T$ is 0, or if $T$ is a write-of-0 node, then $P_{III}(T, 1)$ contains no critical section nodes (so $T$ is normal). If $T$ is a write-of-1, then the register value of $T$ (i.e., the value prior to the write of $T$) is either 0 or 2. It thus suffices to prove the lemma for the case that the register value of $T$ is 2 or that $T$ is a write-of-2. So assume that the Lemma does not hold for such a case.

Under this assumption, the set of edges $E$ of III that abut on nodes of register value 2 and for which $P_{III}(E, 1)$ contains a critical section node, is dense in III (see Definition 4.14(4)). We may thus find such an edge $E$ that is cofinal with the write edge of $\mathsf{V}$ obtained in Proposition 4.10. This clearly contradicts the mutual exclusion property.

**Lemma 4.13** *If $T$ is a write-of-2 node of III, then $P_{III}(T, 0)$ contains a write-of-value $\neq 2$.*

*Proof.* Let $T$ be a write-of-2 of III. Let $\mathsf{W}$ be the write node of II in $P_{II}(\mathsf{S}, 0)$ that is is cofinal with all nodes of III, and in particular with $T$ (Proposition 4.2). We know by Corollary 4.9 that $P_{II}(\mathsf{W}, 2)$ stabilizes at 0.

Suppose (to get a contradiction) that $T$ is the only writes in $P_{III}(T, 0)$. Then a system-execution results in which II reads 2, executes $P_{II}(\mathsf{W}, 2)$ and never changes the stable value of 0, and III reads 0 and never changes the stable value of 2. It can be seen that this contradicts the lockout-freedom, because if both processes arrive to critical section nodes, it is possible to make these nodes concurrent. Thus $P_{III}(T, 0)$ contains a write-of 1 or 0.

## 4.1 The contradiction

Now we can state the main lemma which will bring the contradiction and proof of the theorem.

**Lemma 4.14** *III contains a write-of-2 node $T$ such that $LP_{III}(T,0)$ contains a critical-section node $X$ (so there are no write nodes in between $T$ and $X$ in the limited path, and hence $X$ has register value 2).*

*Proof.* We will define a system-execution in which III executes a path $Q$ which is the concatenation of finite paths $Q_1, \ldots$. For odd $i$'s, $Q_i$ contains no critical-section node, and for even $i$'s, $Q_i$ has the form $LP_{III}(T,0)$ where $T$ is a write-of-2 node. Since $Q$ necessarily contains a critical-section node, the Lemma will be proven.

The path $Q$ includes a sequence $T_i$ of write nodes of III, where $T_{2i}$ is a write-of-2 and $T_{2i+1}$ is a write-of-(0 or 1), defined by induction as follows (see figure 2). Let $T_1$ be the initial write-of-0 by III.

If $T_j$ is a write-of-value $\neq 2$, then $P_{III}(T_j,1)$ contains no entry to a critical-section node (by normality—see Corollary 4.12) but, by Proposition 4.8, does contain a write-of-2. We let $T_{j+1}$ be the first such write.

If $T_j$ is a write-of-2, then we know by Lemma 4.13 that $P_{III}(T_j,0)$ contains a write-of-value $\neq 2$, and we let $T_{j+1}$ be the first such write.

*figure 2*

Let $Q$ be the resulting path of III that contains all the $T_i$'s. We are going to show that there is a system-execution in which $Q$ is realized. That is, II can supply the needed values of 0's and 1's. For this we define a path in II which includes a sequence of write-of-($i$ *mod* 2) nodes, $\mathsf{V}_i$. The edge of $\mathsf{V}_i$ executes and supplies the value needed for the read(s) that take $T_i$ to $T_{i+1}$.

$\mathsf{V}_1$ is a write-of-1 node in $P_{II}(\mathsf{S},0)$ (it exists by the definition of $\mathsf{S}$ and the value 1). If $\mathsf{V}_j$ is defined, then the passage to $\mathsf{V}_{j+1}$ is done by reading the value written by $T_{j+1}$. In case $j$ is even (then $T_{j+1}$ is a write-of-(0 or 1)) use Proposition 4.7(1) or (2) to find a write-of-1 nodes in $P_{II}(\mathsf{V}_j,0)$ or in $P_{II}(V_j,1)$. (Any write of II is $START$-cofinal by Lemma 4.5.) In case $j$ is odd, $\mathsf{V}_j$ is a write-of-1. Let II read the value 2 from $T_{j+1}$, and define the write-of-0 $\mathsf{V}_{j+1}$ by Corollary 4.9.

Since III cannot be locked-out, there must be a critical-section node in $Q$. As $P_{III}(T_j,1)$ contains no critical-section node by normality of $T_j$, such a critical-section node must be in $P_{III}(T_{2j},0)$ for some $j$, and this proves the lemma. ∎

Lemma 4.14 thus shows that the set of nodes defined in 4.15(5) (for $w=2, v=0$) is dense.

We are now ready for the final contradiction. Let $\mathsf{V}$ be a write by II such that $P_{II}(\mathsf{V},2)$ contains a critical-section node (by Proposition 4.10). By Theorem 2.12, $\mathsf{V}$ is cofinal with a write node having the properties of Lemma

4.14. So let $T$ be a write-of-2 node of III such that $LP_{III}(T,0)$ contains an entry to a critical-section node and such that $\mathsf{V}$ and $T$ are cofinal. Stretch a little the interval of the edge of $\mathsf{V}$ just to leave enough time for III to read the value 0 from within that interval of $\mathsf{V}$ and to reach its critical-section state. Now awaken II and let it read the 2 of $T$ so that it too enters a critical section state. *This contradiction to the mutual-exclusion property proves our impossibility Theorem.* ∎

We will define now the finite collection $\mathcal{P}$ of subsets of automaton III that was used in the proof.

**Definition 4.15 The collection $\mathcal{P}$ consists of the following subsets of nodes in III**. *Let $v_0$ be a value of the register of II, and $(w_0, w_1)$ be two values of the register of III. We say that $X$ is a START node of III for the values $(v_0, w_0, w_1)$ iff $X$ is an external node with register value $w_0$, and the path $P_{III}(X, v_0)$ contains infinitely many external nodes of register value $w_0$ and critical-section nodes of register value $w_1$.*

1. *For any $(v_0, w_0, w_1)$ as above put in $\mathcal{P}$ the set of all START nodes $X$ in III for these values.*
2. *For any value $w$ of the register of III, and value $v$ of the register of II, put in $\mathcal{P}$ the set of all critical-section nodes, $E$, of III with register value $w$ such that $P_{III}(E, v)$ contains infinitely many critical-section nodes of value $w$.*
3. *For any possible value $v$ of the register of III, the set of all write-of-v nodes of III is in $\mathcal{P}$.*
4. *For any values $v$ and $w$ of the registers of II and III, put in $\mathcal{P}$ the set of all nodes, $T$, of III that have register value $w$ and such that $P_{III}(T, v)$ contains a critical-section node.*
5. *For any values $v$ and $w$ of the registers of II and III, put in $\mathcal{P}$ the set of all write-of-w nodes, $T$, of III such that $LP_{III}(T, v)$ contains a critical-section node.*

The automaton III referred to in this definition is the original automaton with which the proof begins; after the assignment of new starting nodes, a new automaton was obtained which is a final part of the original, i.e., if $A \leq_{acc} B$ and $A$ is in the new automaton, then $B$ is there as well. It follows that all the definitions in 4.15 are absolute: For any definition $\phi(x)$ given there and node $A$ in the new automaton, $\phi(A)$ holds in the original automaton iff it holds in the new automaton. Thus it does not really matter for the proof if we refer to the original III or the new III in dealing with the sets defined here; and this is also the reason why the same name was used for both automata.

# 5 6 values protocols

Having shown that there can be no solution to the critical-section problem for two processes that use regular registers and only five values, we now turn to positive results when six values are allowed. Two solutions are given: the first uses 2 and 4 values in the two regular registers and the second uses 3 values in each register. (In fact, for the mutual exclusion property, it suffices that the registers are safe (see Definition 2.4). For boolean registers, safeness is equivalent to regularity, but for registers of greater size safeness is a weeker property.) Let us first give a description and correctness proof for the 2-4 values solution.

## 5.1 2-4 regular values protocol

The protocol is given, in figure 3, as automata for the two process called **II** and **IV** which carry 2 and 4 values respectively in their regular registers.

*figure 3*

**Proposition 5.1** *The mutual-exclusion property holds.*

**Proof.** Suppose a system execution of this protocol, and let $CS^{\mathbf{II}}, CS^{\mathbf{IV}}$ be two critical-section events in the first and the second processes. We are going to prove that either $CS^{\mathbf{II}} \longrightarrow CS^{\mathbf{IV}}$ or $CS^{\mathbf{IV}} \longrightarrow CS^{\mathbf{II}}$. This obviously is the mutual-exclusion property. Let $\beta$ be the last write-of-1 done by **II** (at edge $b$) before $CS^{\mathbf{II}}$; and let $\gamma$ be that last read of 0 or 2 done by **II** (at edge $c$) before $CS^{\mathbf{II}}$. So $\beta \longrightarrow \gamma \longrightarrow CS^{\mathbf{II}}$. Observe that **IV** enters the critical-section only after reading 0. Let $R$ be that read-of-zero event done by **IV** just before $CS^{\mathbf{IV}}$. Comparing $R$ and $\beta$, there are two possibilities: $\beta \longrightarrow R$ and $R \dashrightarrow \beta$.

Assume first $\beta \longrightarrow R$. Since $\beta$ is a write-of-1 and $R$ a read of 0, it follows that there is another write, $w$, done by **II** after $\beta$ such that $w \dashrightarrow R$. We have $\beta \longrightarrow CS^{\mathbf{II}} \longrightarrow w \dashrightarrow R \longrightarrow CS^{\mathbf{IV}}$. Thus $CS^{\mathbf{II}} \longrightarrow CS^{\mathbf{IV}}$. (See Definition 2.3(a).)

Now suppose that $R \dashrightarrow \beta$. Let $W$ be the last write by **IV** before $R$; so $W$ is either a write of 1 or of 3 done at edges $A$ or $E$. We have $W \longrightarrow R \dashrightarrow \beta \longrightarrow \gamma$. Hence $W \longrightarrow \gamma$. Yet $\gamma$ is a read of 0 or 2; hence there must be some write $V$ by **IV** such that $CS^{\mathbf{IV}} \longrightarrow V \dashrightarrow \gamma$, and so $CS^{\mathbf{IV}} \longrightarrow CS^{\mathbf{II}}$. (Observe that only the safeness of the registers was used for the mutual-exclusion property.)

**Proposition 5.2** *The lockout-freedom property holds.*

**Proof.** Given a system execution, we have to show that any beginning of the protocol by any process is followed by an entry to the critical-section and then to the external state. The proof is carried under the assumption that any event is terminating (i.e., of finite duration) —except possibly the external states $s$ or $S$ which may correspond to nonterminating events.

First we prove that **II** is never locked out. There are only two places which may cause any trouble: these are the read loops in $a'$ and $c'$. If **II** is caught in a never ending reading loop at $a'$, then the value of its register stabilizes at 0. Thus, from the end of the last write-of-0 by **II** , any read by **IV** obtains the value 0. Looking at automaton **IV** next, we see that it either stays forever in an external state or else it reaches the critical section infinitely many times by reading the 0 of **II**. In the second case, **IV** passes only through $S, A, B, CS, F$; thus only the values 0 and 1 are written by **IV** after a certain time. This contradicts the fact that **II** is only reading 2 and 3. A similar reasoning shows that **II** is never locked out at $c'$. This proves the lockout-freedom for **II** in any system-execution.

Next we turn to **IV**. A look at its protocol shows that there are only two kinds of cycles in its automaton which could possibly lockout **IV**. The read loop at $D'$ and the $(C, (D')^*, D, E, B')^*$ loop. If **IV** were locked at $D'$ in some system-execution, then the value 2 is present forever to **II**, and thus **II** never crosses edge $a$ after the stabilization, and is neither in a nonterminal external state. That is, II is locked-out, in contradiction to what we have just proved. Now the longer loop of **IV** brings an alternation of writes of 2 and 3. Thus from some point on, **II** is able only to read 2's and 3's. Again this contradicts the lockout-freedom of **II**.

Observe that if **IV** is active forever, i.e. no nonterminating external states of **IV** are allowed, then the value 0 of **IV** is not necessary: The 2-3 protocol resulting by abolishing the $F$ edge of **IV** (and connecting the $CS$ edge directly to the external node $S$) satisfies the lockout freedom and the mutual-exclusion properties.

## 5.2 The 3-3 value protocol

The protocol is given for two processes $P_0$ and $P_1$ having each a 3-valued regular register.

*figure 4*

For the mutual-exclusion property it suffices to observe that for both $P_0$ and $P_1$ an entry to the $CS$ node follows a last write-of-1 and a successful read of either 0 or 2. The argument can easily be completed. Again, only the safeness of the registers is needed for the mutual-exclusion property.

The lockout-freedom for $P_0$ is argued by considering the two loops, at $b'$ and $d'$. For example, if $P_0$ is locked out at $d'$, then $P_1$ would finally only be able to read 1's. Now any node of $P_1$ leads, by reads of 1's, to edge $E'$ and then $P_1$ loops forever at $E'$. Thus, if the value of $P_0$ is fixed at 1, then $P_1$ is finally either eternally in its external state or else is stuck at $E'$, contradicting the assumption that $P_0$ only reads 1's. The argument for $b'$ is easy too.

The lockout-freedom for $P_1$ is done by analyzing the three possible causes for lockout: the loop of reads of 2 at $A'$, the reads of 1 at $E'$, and the loop formed by the edges $D, (E')^*, E, B, C$. If $P_1$ were caught at $A'$, then $P_0$ reads only 0's and hence from some point on never reaches the only node of write-of-2. Thus $P_1$ would finally read 0 or 1. Assume $P_1$ is locked at $E'$, eternally reading 1's. Then $P_0$ only reads 2, and an easy contradiction to the lockout-freedom of $P_0$ is derived. Likewise, an infinite loop through write edges $D$ and $B$ would result in $P_0$ reading only 2's and 1's but never 0. Thus (from some point on) $P_0$ cannot cross the wait for 0 read at $b'$. This would contradict the lockout freedom of $P_0$.

Here too it is possible to get a 2-3 protocol for regular registers, but now for the case that the process with the two-valued register is never eternally in its external state. This protocol is obtained by removing the write-of-zero node and edge of $P_0$ and connecting the $CS$ edge directly to the external node.

# 6 Conclusion

The positive and negative results of this paper are represented in the following chart:

| *Process assumptions* | *Register assumptions* | *Results* | Reference |
|---|---|---|---|
| *Both are active* | (2,2) atomic | yes | easy |
| | (2,2) atomic; regular | NO | 3.2 |
| *$X$ may sleep; $Y$ is active* | (2,2) atomic | YES | 3.1 |
| | (2,3) regular | YES | 5.1 |
| | (3,2) regular | YES | 5.2 |
| *Both may sleep* | (2,2) | no | easy |
| | (2,3) atomic; regular | yes | Huddleston |
| | (2,3) regular; atomic | NO | 4 |
| | (2,4) both regular | YES | 5.1 |
| | (3,3) both regular | YES | 5.2 |

The main result of this paper is a proof that there is no solution to the critical-section problem for $n$ processes that use regular registers when the total sizes of the registers is $< 3n$. The difficulty is in the case $n = 2$, and we proved that there is no solution to the critical-section problem for two processes that use a 2-valued regular register for one process and a 3-valued atomic register for the other process. This result shows the necessity of the assumed atomicity of the boolean register in Huddleston's protocol (mentioned in the Preface).

Initially, we thought that this impossibility result is expected and should be easy to prove. We argued informally as follows: The reads of the boolean regular register that are done while the register is being written convey no information at all, since any sequence of values, 0 as well as 1, can be returned. As the reader of the boolean register has no way to tell whether that register is being written or not, no information can be passed and no solution to the critical-section problem is possible under these assumptions. However a proof of this has resisted our efforts for a while. We now know that this informal argument is not valid since ($a$) a 2-4 regular register protocol does exist, and ($b$) if the assumption that any process may sleep forever in an external state is dropped, then regular-register solutions of total size 5 do exist. Indeed it was somewhat surprising to find that if the process owning the boolean regular register promises never to sleep, then a protocol (the variant in Section 5.2) for 2 and 3-valued regular registers solves the critical-section problem. Similarly, the variant given in Section 5.1 for 2 and 3-valued regular registers solves the critical-section problem under the assumption that the process owning the 3-valued register accesses the protocol infinitely often.

The fact that our impossibility result is not immune to small changes in the assumptions, may explain why it led us so deep into the structure of the automata involved. There are other 'impossibility' proofs in the literature but ours has a distinctive character in that we have to poke much deeper into the states and nodes of the assumed protocol-automata in order to derive a contradiction. The classical impossibility result of Fisher, Lynch and Paterson [1983] is looking into all possible scenarios, but at some distance from the automata themselves. Their approach is certainly better—when possible, but our result is probably the first example where general considerations do not suffice and a detailed analysis is necessary.

Similarly, the pioneering analysis made in Burns, Jackson, Lynch, Fischer and Peterson [1982] is relying on partitioning the set of states into 4 classes: the *remainder, trying, critical* and *exit* regions. They have no need there to look into the detailed structure of the assumed protocol. These authors deal with test-and-set registers, and they have asked the general question to which we contribute here a partial answer: *What are the minimal data-type requirements to solve the critical-section problem under different assumptions on the properties of the registers used* (and, we may add, different assumptions on the behavior of the processes—such as whether nonterminating external states are allowed).

The difference between our proof on the one hand and the impossibility result of Fisher et al. and that of Burns et al. on the other hand, is also revealed by the fact that their proofs work for infinite as well as for finite automata with no difference in the proof itself. We too use infinite automata to represent protocols and thus prove that even infinite protocols won't solve the 2 and 3 valued regular register critical-section problem. But our proof was complicated to some extent because of this infinity assumption. We had to use a special 'diagonalizing' argument in order to take care of the infinite case.

Whereas for the case $n = 2$ our results are the best possible, the question remains open for $n > 2$ many processes. Among the works published for $n$ processes, we find Dijkstra [1974] which primarily deals with the question of stabilization, but his protocols are also a solution to the critical-section problem when the processes are assumed to be active for ever. He needs atomic registers of total size $3n$, and if the registers are assumed to be regular, then the protocol fails. For the full critical-section problem, when processes may go out and sleep forever in an external state, Peterson's [1983] algorithm works for $n$, 4-valued, regular registers. Thus the first open problem is the critical-section problem for 3 processes with regular registers: There is a protocol with a total register size of 12, and we only know that there is no such protocol for $\leq 8$ values.

# References


U. Abraham, S. Ben David and M. Magidor

1990 *On Global-Time and Inter-process Communication.* BCS-FACS Workshop on SEMANTICS FOR CONCURRENCY, M. Z. Kwiatkowska et al. (Eds), pp.311-323.

S. Ben David

1988 *The global-time assumption and semantics for distributive systems.* in proceeding of PODC 88.

J. E. Burns, P. Jackson, N. A. Lynch, M. J. Fischer and G. L. Peterson

1982 *Data Requirements for Implementation of N-Process Mutual Exclusion Using a Single Shared Variable.* J. of the ACM, Vol.29, pp.183-205.

E.W Dijkstra

1965 *Co–operating sequential processes.* in Programming languages. F. Genuys (Eds); Academic Press, New York, pp. 43-112.

E.W Dijkstra

1974 *Self-stabilizing systems in spite of distributed control.* Comm. ACM, 17(11), 643-644.

M.J. Fisher, N.A. Lynch, M.S. Paterson

1983 *Impossibility of Distributed Consensus with One Faulty Process.* Pro. of the ACM II Sym. of Prin. of Database Sys.

L. Lamport

1986 *The mutual exclusion problem: Part I: A theory of interprocess communication; Part II: Statements and solutions.* Journal of the ACM, Vol. 33, pp. 313 - 348.

L. Lamport

1986a *On Interprocess Communication, Part I: Basic formalism, Part II: Algorithms.* Distributed Computing, Vol. 1, pp. 77 - 101.

G.L. Peterson

1983 *A new solution to Lamport's concurrent programming problem using small shared variables.* ACM transaction on Programming languages and systems, Vol. 5, pp. 56- 65.

G.L. Peterson and M.J. Fisher

1977 *Economical solutions for the critical section problem in a distributed system.* Conference Record of the Ninth Annual ACM Symposium on Theory of Computing, Boulder; Colo., pp. 91 - 97.

V. Pratt

1986 *Modelling concurrency with partial orders.* Int. J. of Parallel Programming, 15, pp. 33-71.