Constant RMR Transformation to Augment Reader-Writer Locks with Atomic Upgrade/Downgrade Support

Jake S. Leichtling
Dartmouth College

Constant RMR Transformation to Augment Reader-Writer Locks with Atomic Upgrade/Downgrade Support

Jake Stern Leichtling

Thesis Advisor: Prasad Jayanti

With significant contributions from Michael Diamond.

May 29, 2014

Abstract

The reader-writer problem [1] seeks to provide a lock that protects some critical section of code for two classes of processes: *readers* and *writers*. Multiple readers can have access to the critical section simultaneously, but only one writer can have access to the critical section to the exclusion of all other processes. The difficulties in solving the reader-writer problem lie not only in developing a correct and efficient algorithm, but also in rigorously formulating the desirable properties for such an algorithm to have. Bhatt and Jayanti accomplished both of these tasks for several priority variants of the standard reader-writer problem [2][3]. Diamond and Jayanti subsequently formulated the notions of *upgrading* and *downgrading*, in which a reader can attempt to become a writer or a writer can become a reader, respectively, while in the critical section of the lock [4]. They presented an algorithm for a reader-writer lock that supports upgrade/downgrade while giving readers priority over writers in accessing the critical section (the reader-priority variant). In this paper, we formulate the desirable properties of a reader-writer lock which supports upgrade/downgrade as atomic primitives. Furthermore, we propose an algorithm that transforms a standard reader-writer lock of one of several priority variants into a reader-writer lock of the same priority variant that supports upgrade/downgrade as atomic primitives.

Contents

List of Figures

1 Introduction

1.1 The Reader-Writer Problem

Consider a distributed system in which multiple processes are computing concurrently and communicating with each other using atomic shared variables. These processes are asynchronous, meaning that the rate at which each process executes its code is variable and unrelated to the rate of execution of other processes. In such a system, there are frequently resources, such as elements of hardware or segments of memory, that are shared amongst the processes in the system. A certain class of these shared resources requires exclusive access by a single process at any given time—for example, a printer. For such a resource, a protocol is needed to synchronize concurrent attempts by multiple processes to access the resource, granting access to only one process at a time. Such resources give rise to the *mutual exclusion problem*, a fundamental problem in distributed computing first formulated by Dijkstra [5].

Subsequently, other classes of resources that require different access constraints have been considered. One such class contains resources for which a process can either request reader access or writer access—for example, a buffer. For such a resource, a process with writer access must have sole access to the resource, since otherwise, a concurrent reader may read an inconsistent state of the buffer or a concurrent writer may cause the buffer to be in an inconsistent state once both writers have finished. However, it is not necessary that a process with reader access have sole access to the resource, since concurrent readers will not interfere with one another. Furthermore, the efficiency of the system can be increased by allowing processes to concurrently hold reader access to the resource, making the mutual exclusion problem unsuited to this class of resources. Such resources instead give rise to the *reader-writer problem*, a widely known variant of the mutual exclusion that was first formulated by Courtois, Heymans, and Parnas [1].

1.2 Upgrade/Downgrade Support

Within an asynchronous distributed system, a process p, having gained reader access to a shared buffer b, might then decide that it would like to write to b. Ideally, p would not have to relinquish reader access to b and then request writer access, and instead p could request to upgrade its status to a writer. Of course, p's attempt to upgrade must not unequivocally succeed, since there may be another reader with access to b at this time. However, if p alone has access to b and there is no other process waiting to access b, then there is no reason that p should be disallowed from upgrading to a writer.

In a parallel scenario, a process p may have writer access to b, but, having finished writing what it desired to, now wishes to simply read the contents of b. Ideally, p would not have to relinquish writer access and then request reader access, and instead p could simply downgrade its status to a reader, thus allowing other waiting readers to access b while p sticks around. This downgrade should always be available to p, since there is no risk that p's presence as a reader after the downgrade will clash with the presence of a writer, as p alone had access to b beforehand.

These scenarios motivate the desire for a reader-writer synchronization protocol that offers upgrade/downgrade capabilities, with regulations on when an upgrade can succeed. Additionally, it is desirable to have atomic upgrade and downgrade primitives so that the system using such a protocol experiences well-defined upgrade and downgrade behavior, without risk of race conditions. As we will describe later, having atomic upgrade and downgrading primitives simplifies the formulation of the desired properties of a reader-writer lock, making the notion of a process in the midst of upgrading or downgrading irrelevant. The focus of this paper is a transformation that, given a reader-writer lock, enhances it with upgrade/downgrade support in the form of atomic upgrade and downgrade primitives.

1.3 Remote Memory References and the Cache-Coherent Model

The remote memory reference (RMR) complexity of a reader-writer lock $\mathcal{L}$ describes how the number of RMRs made by each process relates to the total number of processes in $\mathcal{L}$. An RMR is related to the access of a shared variable, but the specific definition is dependent on the model of concurrent computation being considered.

For reasons described by Bhatt and Jayanti [2], this paper is concerned with the cache-coherent (CC) model, in which each process keeps a local cache of the values of shared variables. If a process reads a shared variable that it has previously read, and that shared variable has not been updated since the previous read, then the process reads from its local cache. If a process reads a shared variable that it has never read before or that has been updated since the most recent read, then the process must refresh its cached copy and incur an RMR. Any time a process updates a shared variable, it incurs an RMR. We can then formally define a remote memory reference:

Definition 1 A step (C, p, C') incurs a *remote memory reference* if in the step, p performs an update operation on a shared variable or a read operation on a shared variable not in its cache.

As a result of the CC model, a process that is spinning on a shared variable, i.e. repeatedly reading it until its value changes to some desirable value, may incur only a small number of RMRs, even if it reads the shared variable many times.

When concurrent algorithms are implemented in distributed systems, an RMR can take many orders of magnitude more time than accessing local memory. The system may consist of many computers communicating over a network, or threads executing in parallel on different cores of the same processor die; In either case, a process executes most efficiently when it interacts with its own local data, rather than uncached shared data.

Therefore, in the study of concurrent algorithms, it is desirable to devise algorithms that minimizes RMR complexity. This paper presents a transformation that augments a reader-writer lock with upgrade/downgrade support and has a constant RMR complexity ($O(1)$) on its own, i.e. not including the reader-writer lock which it transforms. Specifically, if the reader-writer lock which it augments has constant RMR complexity $O(s(n))$, where n is the number of processes in the system, then the resulting enhanced lock has RMR complexity $O(s(n))$ as well.

1.4 Previous Work

In their seminal formulation of the reader-writer problem, Courtois, Heymans, and Parnas studied two variants that naturally arise when considering which process's turn it is to gain access—one in which readers have priority over writers, and one in which writers have priority over readers [1]. More recently, Bhatt and Jayanti rigorously specified the core desired properties for all reader-writer locks; formulated additional specifications for reader-priority, writer-priority, and starvation-freedom reader-writer locks; and presented constant RMR complexity algorithms for each of these three variants [2].

Subsequently, Diamond and Jayanti rigorously specified the desired properties of a reader-writer lock supporting upgrade/downgrade and presented a constant RMR complexity algorithm for a reader-priority reader-writer lock supporting upgrade/downgrade [4]. In their paper, upgrade and downgrade were not specified as atomic primitives, and the upgrade and downgrade procedures in the presented algorithm were not linearizable. As such, Diamond and Jayanti altered the original specification of reader-priority reader-writer locks given by Bhatt and Jayanti to incorporate the notion of a process in the course of upgrading or downgrading.

In this paper, we specify upgrade and downgrade as atomic primitives and maintain the original specification of reader-writer locks given by Bhatt and Jayanti. The upgrade and downgrade procedures in the algorithm we present are linearizable, such that the notion of a process in the course of upgrading or downgrading is unnecessary in the specification of desired properties for the reader-writer lock.

2 The Model

The environment consists of a set of processes with distinct identities. Each process has a set of local variables accessible only by that process. Additionally, there is a set of shared variables that all processes can access. The transformation presented in this paper requires the following atomic operations on these shared variables: *read*, *write*, *fetch&add*, *compare&swap*. The *read* and *write* operations are self-explanatory. The precise definitions of the *fetch&add* (F&A) and *compare&swap* (CAS) operations are provided in Figure 1.

- F&A(V, b) behaves as follows, where V is a shared numerical variable and b is a number: If V's current value is a, V is assigned $a + b$ and a is returned.
- CAS(X, u, v) behaves as follows, where X is a shared variable, and u and v are values of the type of X: If X's current value is u, X is assigned to v and TRUE is returned; otherwise, X is unchanged and FALSE is returned.

Figure 1: Definitions of the operations F&A and CAS as defined by Bhatt and Jayanti in their paper [2].

In addition to local variables, each process also has a program counter which indicates the next instruction that the process will execute. The *local state* of a process is given by the values of its local variables and its program counter. The *configuration* of the system as a whole is given by the local state of each process and the values of the shared variables. An *algorithm* specifies a program for each process and an initial configuration. The transformation given in this paper is an algorithm that makes use of an input algorithm.

A *step* is a triple (C, p, C'), where C is a configuration, p is a process, and C' is the configuration that results from C after p executes the instruction at its program counter. We call C the start configuration and C' the end configuration, and we say that p has taken a step. A *run from a configuration* C_0 is a list σ of steps such that the start configuration of the first step in σ is C_0, and for each pair of consecutive steps $(C_a, p, C_b), (C_c, p', C_d) \in \sigma$, the end configuration of the first step in the pair is the start configuration of the next, i.e. $C_b = C_c$. A *run* is a run from the initial configuration of the algorithm. A configuration C is *reachable* if C is the initial configuration or if there exists a finite run σ such that C is the end configuration of some step in σ. We assign a time t to each step s in a run σ such that for steps s and s', $t(s) < t(s')$ if and only if s occurs before s' in σ. We say that a process p *crashes* in an infinite run σ if there exists an end configuration C in σ such that p does not take any steps after C.

3 Specification of the Reader-Writer Problem with Upgrade/Downgrade

An algorithm for the reader-writer problem with upgrade and downgrade is broken into several sections of code: the *Remainder section*, the *Try section*, the *Critical section* (CS), and the *Exit* *section.* While in the Remainder section, a process does not execute any code in the algorithm. In the Try section, a process tries to obtain either read or write access to the CS. The specific Try section procedure that a process executes determines what type of access it is initially granted when admitted to the CS. While in the CS, a process that has read access may attempt to upgrade to gain write access, and a process that has write access may downgrade to relinquish its write privileges. In the Exit section, a process relinquishes all access. Note that different types of processes execute different code in each section.

Each process's code consists of a loop where it executes the Try section, optionally attempts to upgrade and downgrades an arbitrary number of times, and then executes the Exit section before beginning the loop again. We define a process to be in the Remainder section when its program counter is on the first line of the Try section. We define a process to be in the CS when its program counter is on a line of an upgrade or downgrade procedure or on the first line of the Exit section. For convenience, when a process enters the CS or finishes executing an upgrade or downgrade procedure, we say its program counter points to the next section of code that it will execute, that is, either the Exit section section or the beginning of an upgrade or downgrade procedure. Similarly, when a process finishes the Exit section, we say its program counter points to the first line of the Try section.

We define an *attempt* at the CS as an execution of the Try section and a subsequent execution of the Exit section (with an arbitrary number of executions of upgrade and downgrade procedures while in the CS) by a single process. More formally, an attempt A by a process p begins at some time t when p executes the first statement of the Try section and concludes at the earliest time $t' > t$ that p has completed the Exit section. Furthermore, we define a *read attempt* as an attempt by a process that executes the Try section with the intent to gain reader access to the CS. In such a read attempt, the process is referred to as an *original reader*, denoting the fact that while its type of access to the CS may change via upgrade and downgrade, it initially executed the Try section code to gain reader access. Similarly, we define a *write attempt* as an attempt by a process that executes the Try section with the intent to gain writer access to the CS. In such a write attempt, the process is referred to as an *original writer*, denoting the fact that while its type of access to the CS may change via downgrade and upgrade, it initially executed the Try section code to gain writer access.

In an attempt by a process p, p enters the Try section as a *reader* if it is an original reader and as a *writer* if it is an original writer, and p initially has the corresponding privileges when it is admitted into the CS. While in the CS, p can downgrade to a reader if it is currently a writer, or attempt to upgrade to a writer if it is currently a reader. An *upgrade attempt*, i.e. the execution of the upgrade procedure by a reader in the CS, will not always be successful: For example, if there are multiple readers in the CS, a process attempting to upgrade must not be granted writer access. Therefore, an upgrade attempt will yield a result of *success* or *failure*. In contrast, the execution of a downgrade procedure will always be successful, as there is no potential danger in trading a process reader access for writer access. In this paper, upgrade and downgrade procedures are seen as atomic primitives, such that a reader that successfully upgrades becomes a writer at a distinct step during the upgrade attempt, and a downgrading writer becomes a reader at a distinct step during the downgrade.

Processes can execute upgrade and downgrade procedures an arbitrary number of times while they are in the CS, so we keep track of both the original status and current status of each process. To reiterate, in an attempt A by a process p, the original reader/original writer designation encapsulates the initial status of p throughout A, while the reader/writer designation encapsulates the current status of p at any point in A. A process's original and current statuses are considered a component of its local state. We then place the following restrictions on the executions of upgrade and downgrade procedures:

1. A process in the CS may attempt to upgrade only if it is a reader.
2. A process in the CS may downgrade only if it is a writer.

When discussing the Try section, it is useful to break it down further into a *doorway* and a *waiting room* [6]. The doorway is a contiguous section of straight-line code in which no waiting occurs that begins the Try section. The waiting room is all code that follows the doorway within the Try section. The concept of a doorway allows us to formulate the notion of one process requesting access to the CS before another one:

Definition 2 An attempt A by a process p *doorway precedes* an attempt A' by a process p' if p completes the execution of its doorway in A before p' completes the execution of its doorway in A'.

Two attempts A and A' are considered to be *doorway concurrent* if neither attempt doorway precedes the other.

We say a process p doorway precedes a process p' at some time t if the current attempt of p doorway precedes the current attempt of p' or the subsequent attempt of p' if p' is in the Remainder section at time t.

We say a process p is doorway concurrent with a process p' if neither process doorway precedes the other.

It can also be useful to talk about a process being guaranteed to have access to the CS granted, regardless of the actions of other processes. We formulate this notion with the following definition:

Definition 3 A process p is *enabled* to enter the CS in a configuration C if p is in the Try section in C and there exists some fixed bound b such that for all runs from C, p enters the CS in at most b of its own steps.

3.1 Specification of the Reader-Writer Problem

Listed below are desirable properties for a reader-writer lock as formulated by Bhatt [7]. For brevity, some of the context for the properties has been removed. For justification of the desirability of these properties, see Bhatt's paper.

- (P1) Mutual Exclusion: If a writer is in the CS at any time, then no other process is in the CS at that time.
- (P2) Bounded Exit: There is an integer b such that in every run, each process completes the Exit section in at most b of its own steps.
- (P3) First-Come-First-Served (FCFS) among writers: If w and w' are any two write attempts in a run and w doorway precedes w', then w' does not enter the CS before w.
- (P4) First-In-First-Enabled (FIFE) among readers: Let r and r' be any two read attempts in a run such that r doorway precedes r'. If r' enters the CS before r, then r is enabled to enter the CS at the time r' enters the CS.
- (P5) Concurrent Entering: Informally, if all writers are in the Remainder section, readers should not experience any waiting, i.e. every reader in the Try section should be able to proceed to the CS in a bounded number of its own steps. More precisely, there is an integer b such that, if σ is any run from a reachable configuration such that all writers are in the Remainder section in every configuration in σ, then every read attempt in σ executes at most b steps of the Try section before entering the CS.
- (P6) Livelock-Freedom: If no process crashes in an infinite run, then infinitely many attempts complete in that run.

A reader-writer lock of a given variant will satisfy either *reader-priority*, *writer-priority*, or *starvation-freedom* (SF). The transformation presented in this paper does not apply to writer-priority reader-writer locks, so we do not include a rigorous formulation of writer-priority.

Reader-priority captures the notion that if ever a read attempt requests access to the CS before a write attempt, the read attempt should be granted access first. Additionally, if a read attempt and a write attempt are ever waiting for access at the same time, then the read attempt should be granted access first. We encapsulate these expectations with the *reader-priority relation* $>_{rp}$ described by Bhatt and Jayanti [2], which is a binary relation between the set of read attempts and the set of write attempts in a run:

Definition 4 Let r and w be a read attempt and a write attempt, respectively, in a run. We define $r >_{rp} w$ if:

- r doorway precedes w, or

- There is a time when some reader or writer is in the CS, r is in the waiting room, and w is in the Try section.

A reader-priority reader-writer lock then satisfies the following property:

- (RP1) Reader-Priority: Let r and w be a read attempt and a write attempt, respectively, in a run. If $r >_{rp} w$, then w does not enter the CS before r.

Bhatt and Jayanti describe an additional property that reader-priority reader-writer locks can optionally satisfy [2]. The property encapsulates the notion that given a read attempt r, if, for all write attempts w, w will not enter the CS before r, then r should in fact be enabled to enter the CS.

- (RP2) Unstoppable Reader Property: Let C be any reachable configuration in which some read attempt r is in the waiting room.
  - If a reader is in the CS in C, then r is enabled to enter the CS in C.
  - If no writer is in the CS or the Exit section in C and $r >_{rp} w$ holds for all write attempts w that are in the Try section in C, then r is enabled to enter the CS in C.

Note that with reader-priority, it is not necessarily the case that every process that attempts to access the CS will eventually be granted access. For example, consider the scenario in which a reader r is in the CS and a writer w is in the Try section. Then a new reader r' enters the Try section and executes the doorway while r is still in the CS. Since $r' >_{rp} w$, by (RP1), w will not enter the CS before r'. Now r exits the CS, but r' is still inside of it while a new reader r'' enters the Try section and executes the doorway. This cycle can continue indefinitely, blocking w from ever entering the CS in an infinite run.

To capture the desire to avoid such starvation, Bhatt and Jayanti [2] formulated starvation-freedom, which encapsulates the notion that regardless of priority, every process that attempts to access the CS should eventually be granted access.

- (SF) Starvation-Freedom: If no process crashes in an infinite run, then all attempts complete in that run.

3.2 Additional Properties for Upgrade/Downgrade Support

Now we consider the desirable properties of a reader-writer lock with regards to upgrading and downgrading. Primarily, we consider when an execution of the upgrade section should be successful and when it should be unsuccessful. If another process is in the CS, obviously an upgrade attempt cannot succeed or it would violate mutual exclusion. While upgrade could conceivably succeed while there are readers in the in the waiting room, this may be undesirable in a specification where writers are not given priority. Therefore, in the transformation presented in this paper, we consider a specification in which an upgrade attempt should not succeed when there is a reader in the waiting room. Additionally, an upgrade attempt should succeed if there are no other readers present.

- (P7) Upgradeability: If a process in the CS attempts to upgrade and all readers other than that process are in the Remainder section, then the upgrade attempt will succeed.
- (P8) Non-Upgradeability: If a process in the CS attempts to upgrade and a reader is in the waiting room, then the upgrade attempt will fail.
- (P9) Bounded Upgrade: There is an integer b such that in every run where a process executes an upgrade procedure, it completes the upgrade procedure in at most b of its own steps.
- (P10) Bounded Downgrade: There is an integer b such that in every run where a process executes a downgrade procedure, it completes the downgrade procedure in at most b of its own steps.

Note that (P7) and (P8) are written with an upgrade attempt as an atomic action, while (P9) and (P10) refer to the internal execution of upgrade and downgrade attempts. In a reader-writer lock with upgrade/downgrade support, upgrade and downgrade procedures can be comprised of an arbitrary number of lines of code. However, these procedures must be linearizable such that they can be viewed as atomic primitives, with an upgrade attempt succeeding or failing and a downgrade occurring at a discrete time—the linearization point—during execution of an upgrade or downgrade procedure. It is at this linearization point that a process's current status may change and the constraints of (P7) and (P8) apply.

4 Single-Writer Multi-Reader Transformation to Augment Reader-Writer Locks with Upgrade/Downgrade Support

The algorithm in Figure 2 transforms a single-writer multi-reader reader-writer lock $\mathcal{L}$ into a single-writer reader-writer lock $\mathcal{L}'$ that additionally supports atomic (i.e. linearizable) upgrade and downgrade primitives. A highlight is that the transformation preserves the properties of $\mathcal{L}$, regardless of whether $\mathcal{L}$ is a Starvation-Freedom or a Reader-Priority lock: If $\mathcal{L}$ is a Starvation-Freedom lock, then so is $\mathcal{L}'$; and, if $\mathcal{L}$ is a Reader-Priority lock, then so is $\mathcal{L}'$. We will later comment on extending $\mathcal{L}'$ into a multi-writer lock.

Shared Variables

$RC \in \{0\} \cup \mathbb{Z}^+$, initialized to 0
$US \in \{\textsc{upgrading}, \textsc{upgraded}, \textsc{null}\}$, initialized to NULL
$Permit \in \{\textsc{true}, \textsc{false}\}$, initialized to TRUE
$DowngradingWriter \in \{\textsc{true}, \textsc{false}\}$, initialized to FALSE

Local Variables

$original\text{-}status \in \{\mathrm{R}, \mathrm{W}\}$
$current\text{-}status \in \{r, w, \dot{w}\}$
$status \equiv (original\text{-}status, current\text{-}status)$

```
READER-TRY()
1  F&A($RC$, 1)
2  CAS($US$, UPGRADING, NULL)
3  $\mathcal{L}$.READER-DOORWAY()
4  $\mathcal{L}$.READER-WAITINGROOM()
5  if $US$ == UPGRADED
6      wait until $Permit$ == TRUE
7  wait until $DowngradingWriter$ == FALSE

(R, $r$)-EXIT()
8  F&A($RC$, −1)
9  $\mathcal{L}$.READER-EXIT()

(R, $r$)/(W, $r$)-UPGRADE()
10  if $RC \neq 1$ return FALSE
11  $US$ = UPGRADING
12  if $RC \neq 1$ return FALSE
13  $Permit$ = FALSE
14  return CAS($US$, UPGRADING, UPGRADED)

(R, $\dot{w}$)/(W, $\dot{w}$)-RE-DOWNGRADE()
15  $Permit$ = TRUE

WRITER-TRY()
16  $\mathcal{L}$.WRITER-TRY()

(W, $w$)-EXIT()
17  $\mathcal{L}$.WRITER-EXIT()

(W, $w$)-DOWNGRADE()
18  $DowngradingWriter$ = TRUE
19  F&A($RC$, 1)
20  $\mathcal{L}$.WRITER-EXIT()
21  $DowngradingWriter$ = FALSE

(W, $r$)-EXIT()
22  F&A($RC$, −1)
```

Figure 2: Single-writer multi-reader transformation augmenting a reader-writer lock $\mathcal{L}$ with upgrade/downgrade support.

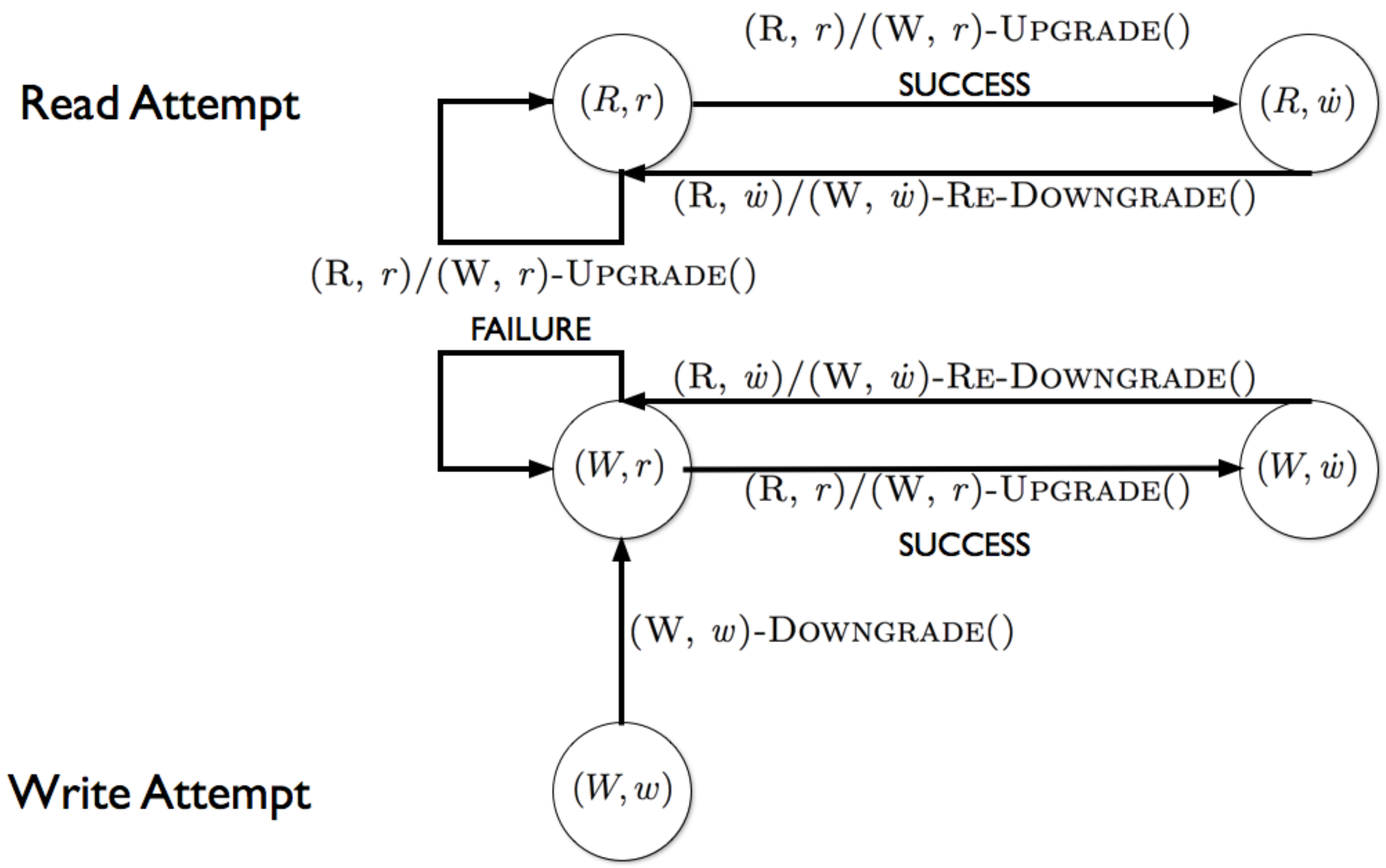

Figure 3: A diagram illustrating how a process's status changes as it upgrades and downgrades, and what upgrade and downgrade procedures are available to it given its status.

4.1 Process Status and Path of Execution

Each process in $\mathcal{L}'$ has an original status that is permanent, which is either R, for read attempts, or W, for write attempts. Since $\mathcal{L}'$ is a single-writer lock, there is only one original writer. We call this original writer $\bar{w}$ (not to be confused with w or $\dot{w}$, which are potential values of the *current-status* local variable). Every process also has a current status, denoting whether it has reader or writer privileges in the CS and what procedures it is allowed to execute. Read attempts begin by executing Reader-Try() with a current status of r, while $\bar{w}$ begins its attempt by executing Writer-Try() with a current status of w.

A process with a current status of r has reader privileges, and, for the sake of the properties satisfied by $\mathcal{L}$ and $\mathcal{L}'$, is considered a *reader*. A process with a current status of w or $\dot{w}$ has writer privileges, and, for the sake of the properties satisfied by $\mathcal{L}$ and $\mathcal{L}'$, is considered a *writer*. In $\mathcal{L}'$ given by Figure 2, the name prefixes of procedures denote the statuses of processes that are allowed to execute them. Note that a process with *current-status* $= \dot{w}$ may not exit directly; rather, it must first downgrade and can then exit with *current-status* $= r$.

Figure 3 demonstrates how a process's status changes as it upgrades and downgrades, and what upgrade and downgrade procedures are available to it given its status. In the figure, only procedures that potentially change the current status of a process are shown. Note that an attempt to upgrade is not guaranteed to succeed, so, upon returning from (R, r)/(W, r)-Upgrade(), a process may have either *current-status* $= \dot{w}$ or *current-status* $= r$.

4.2 Linearization of Upgrade/Downgrade

The upgrade and downgrade procedures—namely, (R, r)/(W, r)-Upgrade(), (R, $\dot{w}$)/(W, $\dot{w}$)-Re-Downgrade(), and (W, w)-Downgrade()—are linearlizable, so they can be considered as atomic primitives. The linearization points are as follows:

- (R, r)/(W, r)-Upgrade() is linearized at one of multiple points, depending on the success or failure of the upgrade attempt by a process p. If the attempt succeeds, then the success is linearized to the time at which it returns true at line 14. If the attempt fails, returning at either line 10 or line 12, then the failure is linearized to the time at which it returns false at line 10 or line 12, respectively. If the attempt fails, returning false at line 14 because the *compare&swap* fails, then the failure is linearized to the earliest time when $PC_p \in \{12, 13, 14\}$ and $\exists q : PC_q = 3$. (We will later show that such a time must exist if the procedure returns false at line 14.)
- (R, $\dot{w}$)/(W, $\dot{w}$)-Re-Downgrade() is linearized at the time line 15 is executed.
- (W, w)-Downgrade() is linearized at the time line 21 is executed.

4.3 Informal Description of the Single-Writer Transformation

In this section, "upgrading reader" denotes a reader currently executing (R, r)/(W, r)-UPGRADE(), "downgrading writer" denotes the original writer $\bar{w}$ currently executing (W, w)-DOWNGRADE(), and "re-downgrading writer" denotes a writer executing (R, $\dot{w}$)/(W, $\dot{w}$)-RE-DOWNGRADE().

In describing the transformation, we sometimes refer to lines of the transformation that are calls to procedures of $\mathcal{L}$—namely, lines 3, 4, 9, 16, 17, and 20. For a line $x \in \{3, 4, 9, 16, 17, 20\}$ of the transformation, x:all refers to the set of instructions that a process will execute in the procedure called at line x.

4.3.1 Overview

The overall idea is as follows. Before being let into the CS, read attempts and write attempts must be let into the CS of the original reader-writer lock $\mathcal{L}$. However, there are additional protections in the Try section of read attempts to ensure that no one enters the CS at an inappropriate time. For example, while a process is upgraded, a read attempt that has been let into the CS of $\mathcal{L}$ will find that US = UPGRADED and $Permit$ = FALSE, and it will busy-wait on $Permit$ until it is set to TRUE when the upgraded process re-downgrades. Additionally, when $\bar{w}$ is downgrading, a read attempt will find that $DowngradingWriter$ = TRUE, and the attempt will busy-wait until it is set to FALSE as the downgrade completes. A process that wishes to upgrade must first check that it is the only reader at the moment, i.e. the count of current readers $RC = 1$, and then it can perform a handshake that, upon succeeding if no readers interrupt it, locks other readers out of the CS by setting US = UPGRADED and $Permit$ = FALSE. When $\bar{w}$ downgrades, it registers itself as a reader by incrementing RC and then executes the Exit section of $\mathcal{L}$, allowing read attempts to enter the CS of $\mathcal{L}$.

4.3.2 Shared Variables and Their Purposes

All the shared variable names start with upper case and the local variable names start with lower case.

$\underline{RC}$: "Reader Count". An integer read/*fetch&add* variable updated by readers and the original writer $\bar{w}$ when it downgrades, and read by upgrading readers. This variable acts as a count of the number of readers around, disallowing readers from upgrading if they are not alone. It is incremented by read attempts at the beginning of the Try section and by $\bar{w}$ when it downgrades, and it is decremented by exiting readers. An upgrade attempt will fail if it reads a value of RC that is greater than 1.

$\underline{US}$: "Upgrade Status". A *compare&swap* variable that takes on a value in {UPGRADING, UPGRADED, NULL} and is updated by read attempts and upgrading readers. When a reader is upgrading, it sets US = UPGRADING at the beginning of its handshake. At the end of the handshake, if US is still UPGRADING, then the upgrade will succeed and US will be changed to UPGRADED. However, an incoming read attempt will *compare&swap* US from UPGRADING to NULL, foiling any upgrading handshake that is in progress.

Permit: A Boolean read/write variable that is read by read attempts, and written by upgrading readers and re-downgrading writers. This variable is used to lock read attempts out of the CS if there is an upgraded writer. It is set to FALSE by the upgrading reader handshake, so if the upgrade succeeds then a read attempt will busy-wait on *Permit* until it is TRUE. This occurs when the upgraded writer re-downgrades.

DowngradingWriter: A Boolean read/write variable that is read by read attempts, and written only by $\bar{w}$ when it downgrades. This variable is used to lock read attempts out of the CS while $\bar{w}$ is in the course of downgrading. It is set to TRUE at the beginning of $\bar{w}$'s downgrade, and then set back to FALSE at the end of the downgrade. If a read attempt tries to enter the CS during this downgrade, it will have to busy-wait on *DowngradingWriter* until $\bar{w}$ finishes downgrading.

4.3.3 Line by Line Commentary

First, we describe the code of a read attempt r in the READER-TRY() procedure
(lines $1, 2, 3$:all, 4:all, $5, 6, 7$). r begins the bounded doorway (lines $1, 2, 3$:all) by incrementing RC to reflect the incoming read attempt (line 1) and performing *compare&swap* on US, possibly changing its value from UPGRADING to NULL in order to foil a concurrent upgrade attempt (line 2). Then r executes the Try section of $\mathcal{L}$ (lines 3:all) and enters the waiting room of $\mathcal{L}$ (lines 4:all). Once r is admitted into the CS of $\mathcal{L}$, it checks if US is UPGRADED (line 5), which would be the case if there were an upgraded writer but does not necessarily imply that there is one. If US is UPGRADED, then r busy-waits on *Permit* until it is TRUE (line 6), which occurs when the upgraded writer re-downgrades (or if there were no upgraded writer to begin with). Just before entering the CS, r busy-waits on *DowngradingWriter* until it is FALSE (line 7), indicating that $\bar{w}$ is not currently downgrading.

Now we describe the code of an upgrading reader r in the (R, r)/(W, r)-UPGRADE() procedure (lines $10, 11, 12, 13, 14$). Note that r may have originally been a writer. r begins upgrading by checking if it is the only reader around, i.e. by ensuring that RC is 1 (line 10). If not, then the upgrade fails and the failure can be linearized at this step. If so, then r proceeds to set US to UPGRADING (line 11), beginning the upgrade handshake. Throughout the remainder of the upgrade attempt, if any new read attempt enters the waiting room, then it certainly changed the value of US to NULL along the way, preventing r's upgrade attempt from succeeding. r then checks that RC is 1 once again (line 12) to ensure that a read attempt did not slip into the waiting room between when r previously checked that it was the lone reader and when it set US to UPGRADING. If r finds that $RC > 1$, then the upgrade fails and the failure can be linearized at this step. If r finds that RC is 1, then it proceeds to change *Permit* to FALSE (line 13), which will lock read attempts out of the CS if the upgrade succeeds. Finally, r performs *compare&swap* on US (line 14), changing it to UPGRADED if it was still UPGRADING. If the *compare&swap* succeeds, then the upgrade succeeds at this step, and the combination of US = UPGRADED and $Permit$ = FALSE locks read attempts out of the CS. If the *compare&swap* fails, then some new read attempt must have changed the value of US to NULL. In this case, the upgrade failure can be linearized to the step in which a read attempt most recently performed *compare&swap* on US in the Try section.

Now we describe the code of a re-downgrading writer p in the (R, $\dot{w}$)/(W, $\dot{w}$)-DOWNGRADE() procedure (line 15). This procedure is only a single line, in which p sets $Permit$ to TRUE. Doing so allows read attempts that were previously locked out of the CS when p was upgraded to enter the CS after p re-downgrades.

Now we describe the code of an exiting original reader r in (R, r)-EXIT() (lines 8, 9). r begins exiting by de-registering as a reader, i.e. decrementing RC (line 8). r then executes the Exit section of $\mathcal{L}$ (lines 9:all) so that $\bar{w}$ will be able to enter the CS once there are no more readers ahead of it "in line".

Now we describe the code of the write attempt $\bar{w}$ in the WRITER-TRY() procedure (lines 16:all). This procedure consists of the Try section of $\mathcal{L}$, so $\bar{w}$ enters the CS as soon as it is admitted into the CS of $\mathcal{L}$.

Now we describe the code of the original writer $\bar{w}$ when it exits—not having downgraded in its attempt—in the (W, w)-EXIT() procedure (lines 17:all). This procedure consists of the Exit section of $\mathcal{L}$, so read attempts can be admitted into the CS of $\mathcal{L}$ once $\bar{w}$ has completed exiting.

Now we describe the code of the original writer $\bar{w}$ when it downgrades in the (W, w)-DOWNGRADE() procedure (lines 18, 19, 20:all, 21). $\bar{w}$ first sets $DowngradingWriter$ to TRUE (line 18) to ensure that read attempts are locked out of the CS for the remainder of the downgrade. $\bar{w}$ then registers itself as a reader by incrementing RC (line 19) and executes the Exit section of $\mathcal{L}$ (lines 20:all) to allow read attempts in the waiting room of $\mathcal{L}$ to enter the CS of $\mathcal{L}$. Finally, $\bar{w}$ unlocks the CS for read attempts by setting $DowngradingWriter$ to FALSE (line 21).

Lastly, we describe the code of the original writer $\bar{w}$ when it exits—having previously downgraded—in the (W, r)-EXIT() procedure (line 22). $\bar{w}$ simply de-registers as a reader, decrementing RC. There is no need for $\bar{w}$ to interact with $\mathcal{L}$ at all, since it previously executed the Exit section of $\mathcal{L}$ when it first downgraded.

5 Proof of Correctness

Theorem 1 (Single-Writer Multi-Reader transformation supporting Upgrade/Downgrade)
The algorithm in Figure 2 transforms a Single-Writer Multi-Reader lock $\mathcal{L}$ satisfying properties (P1), (P2), (P3), (P4), (P5), and (P6) into a Single-Writer Multi-Reader lock $\mathcal{L}'$ that satisfies these same properties and additionally supports Upgrade/Downgrade. If $\mathcal{L}$ satisfies (SF), then $\mathcal{L}'$

does as well. Alternatively, if $\mathcal{L}$ satisfies (RP1), then $\mathcal{L}'$ does as well. If $\mathcal{L}$ satisfies both (RP1) and (RP2), then $\mathcal{L}$ does as well. If the RMR complexity of $\mathcal{L}$ is $O(s(n))$, where n is the number of attempts, then the RMR complexity of $\mathcal{L}'$ is also $O(s(n))$. If the number of shared variables per process used by $\mathcal{L}$ is $O(s'(n))$, where n is the number of attempts, then the number of shared variables per process used by $\mathcal{L}'$ is also $O(s'(n))$.

We will prove the algorithm correct using a set of invariants, $\mathcal{I}$. We will prove that the invariants are correct inductively by proving:

1. $\mathcal{I}$ holds in the initial configuration, and
2. If $\mathcal{I}$ holds in configuration C, then it also holds in any configuration $C.s$ resulting from a step s performed by any process.

5.1 Invariants

A configuration C corresponds to the value of shared variables RC, US, $Permit$, and $DowngradingWriter$, as well as the local state of each process. The state of a process p is given by its program counter, PC_p, and the values of its local variables *original-status* and *current-status*.

5.1.1 Notation Used in the Invariants

Here is some of the notation used in the invariants:

- We denote the value of a local variable y of a process p by $p.y$.
- Since there is only a single original writer, we call it $\bar{w}$. The name of this original-writer $\bar{w}$ should not be confused with the possible values w and $\dot{w}$ of *current-status*.
- PC_p is the program counter of a process p.
- Certain lines of $\mathcal{L}'$ are calls to procedures of $\mathcal{L}$—namely, lines 3, 4, 9, 16, 17, and 20. For a given process p and a line $x \in \{3, 4, 9, 16, 17, 20\}$ in $\mathcal{L}'$, $PC_p = x$ denotes that p has not yet begun executing the procedure called at line x, but will execute the first line of it in its next step. $PC_p \in \{x\text{:all}\}$ denotes that either $PC_p = x$ or p has already begun executing the procedure called at line x but has not yet completed it, i.e. the next instruction that p executes will be part of the procedure called at line x.
- The following are predefined sets used to describe the program counters of processes:
  - CS $= \{8, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20\text{:all}, 21, 22\}$ is the critical section of $\mathcal{L}'$.
  - $\mathcal{L}.\text{CS}_R = \text{CS} \cup \{5, 6, 7, 9\}$ is the critical section of $\mathcal{L}$ that is accessible by original readers.
  - $\mathcal{L}.\text{CS}_W = \{17, 18, 19, 20\}$ is the critical section of $\mathcal{L}$ that is accessible by original writers.

5.1.2 Specification of Invariants

Now we specify the set of invariants $\mathcal{I}$ satisfied by any reachable configuration C of $\mathcal{L}'$. The set $\mathcal{I}$ is comprised of the following invariants:

- $\mathcal{I}_1$: $(\exists p : p.\textit{current-status} = \dot{w}) \rightarrow (\forall q \neq p : PC_q \notin \text{CS} \cup \{7\})$
- $\mathcal{I}_2$: $RC = |\{p : PC_p \in \{2, 3\text{:all}, 4\text{:all}, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 20\text{:all}, 21, 22\}\}|$
- $\mathcal{I}_3$: $(\exists p : p.\textit{current-status} = \dot{w}) \leftrightarrow (\textit{Permit} = \text{FALSE} \wedge \textit{US} = \text{UPGRADED})$
- $\mathcal{I}_4$: $(\exists p : PC_p \in \{11, 12, 13, 14\}) \rightarrow (\forall q \neq p : PC_q \notin \{11, 12, 13, 14\})$
- $\mathcal{I}_5$: $(\textit{US} = \text{UPGRADING} \wedge \exists p : PC_p \in \{13, 14\}) \rightarrow (\forall q \neq p : PC_q \notin \text{CS} \cup \{3\text{:all}, 4\text{:all}, 5, 6, 7\})$
- $\mathcal{I}_6$: $(\exists p : PC_p \in \{12, 13, 14\}) \rightarrow (\textit{US} \in \{\text{UPGRADING}, \text{NULL}\})$
- $\mathcal{I}_7$: $(\exists p : PC_p = 14) \rightarrow (\textit{Permit} = \text{FALSE})$
- $\mathcal{I}_8$: $(\exists p : PC_p = 6 \wedge \textit{Permit} = \text{FALSE}) \rightarrow (\exists q : q.\textit{current-status} = \dot{w})$
- $\mathcal{I}_9$: $(\exists p : PC_p = 6) \rightarrow (\forall q : PC_q \notin \{13, 14\})$
- $\mathcal{I}_{10}$: $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}) \leftrightarrow (\textit{DowngradingWriter} = \text{TRUE})$
- $\mathcal{I}_{11}$: $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}) \rightarrow (\forall p \neq \bar{w} : PC_p \notin CS)$.

5.2 Proof of the Invariants

5.2.1 Auxiliary Lemmas

We now prove some lemmas and a corollary that will be useful in the proof of correctness of the invariants.

Reader Lemma $(p.\textit{current-status} = r) \leftrightarrow (PC_p \in \{1, 2, 3\text{:all}, 4\text{:all}, 5, 6, 7, 8, 9\text{:all}, 10, 11, 12, 13, 14, 22\})$

Proof: An original reader p first executes Reader-Try(), during which $PC_p \in \{1, 2, 3\text{:all}, 4\text{:all}, 5, 6, 7\}$. While in the CS, if a reader p chooses to attempt to upgrade, it executes (R, r)/(W, r)-Upgrade(), during which $PC_p \in \{10, 11, 12, 13, 14\}$. When an original-reader p that is currently a reader wishes to exit the CS, it executes (R, r)-Exit(), during which $PC_p \in \{8, 9\text{:all}\}$. When $\bar{w}$ is currently a reader and wishes to exit the CS, it executes (W, r)-Exit(), during which $PC_{\bar{w}} = 22$. □

Upgraded Writer Lemma $(p.\textit{current-status} = \dot{w}) \leftrightarrow (PC_p = 15)$

Proof: A reader p becomes a writer, obtaining $\textit{current-status} = \dot{w}$, only after successfully executing line 14. The only procedure that p can execute next is (R, $\dot{w}$)/(W, $\dot{w}$)-Re-Downgrade(), so $PC_p = 15$. Once p executes (R, $\dot{w}$)/(W, $\dot{w}$)-Re-Downgrade(), $p.current\text{-}status = r$. □

Writer Lemma $(\bar{w}.current\text{-}status = w) \leftrightarrow (PC_{\bar{w}} \in \{16{:}\text{all}, 17{:}\text{all}, 18, 19, 20{:}\text{all}, 21\})$

Proof: The single original writer $\bar{w}$ first executes Writer-Try(), during which $PC_{\bar{w}} \in \{16{:}\text{all}\}$. While in the CS, if $\bar{w}$ chooses to downgrade, it executes (W, w)-Downgrade(), during which $PC_{\bar{w}} \in \{18, 19, 20{:}\text{all}, 21\}$. If $\bar{w}$ chooses to exit without ever having downgraded, it executes (W, w)-Exit(), during which $PC_{\bar{w}} \in \{17{:}\text{all}\}$. If $\bar{w}$ downgrades, it relinquishes its *current-status* of w forever (although it may upgrade to become a writer with $current\text{-}status = \dot{w}$). □

Lemma 1 $(PC_{\bar{w}} \in \mathcal{L}.\text{CS}_W) \rightarrow (\forall q \neq \bar{w} : PC_q \notin \mathcal{L}.\text{CS}_R \cup \mathcal{L}.\text{CS}_W)$

Proof: $\mathcal{L}$ satisfies Mutual Exclusion. Therefore, if the single writer $\bar{w}$ is in the CS of $\mathcal{L}$, then no other process is in the CS of $\mathcal{L}$. □

Corollary 1 $(PC_{\bar{w}} \in \mathcal{L}.\text{CS}_W) \rightarrow (\forall q : q.current\text{-}status \neq \dot{w})$

Proof: Suppose, for the sake of contradiction, that $(PC_{\bar{w}} \in \mathcal{L}.\text{CS}_W)$ and $(\exists q : q.current\text{-}status = \dot{w})$. From the Upgraded Writer Lemma, we have $PC_q = 15$, so $PC_q \in \mathcal{L}.\text{CS}_R$. This contradicts Lemma 1. □

5.2.2 Inductive Base Case

Lemma 2 *If C_0 is the initial configuration of $\mathcal{L}'$ given in Figure 2, then $\mathcal{I}$ holds in C_0.*

Initially in the algorithm: $\bar{w}.current\text{-}status = w$, $PC_{\bar{w}} = 16$, and $\forall p \neq \bar{w} : PC_p = 1 \wedge p.current\text{-}status = r$. Therefore, $\mathcal{I}_1$, $\mathcal{I}_4$, $\mathcal{I}_5$, $\mathcal{I}_6$, $\mathcal{I}_7$, $\mathcal{I}_8$, $\mathcal{I}_9$, $\mathcal{I}_{10}$, and $\mathcal{I}_{11}$ vacuously hold. Additionally, $RC = 0$ and $US =$, so $\mathcal{I}_2$ and $\mathcal{I}_3$ hold. It follows that $\mathcal{I}$ holds in C_0. □

5.2.3 Induction Step

Lemma 3 *If $\mathcal{I}$ holds in the configuration C, then it also holds in the configuration $C.s$, where $C.s$ is the configuration after some process p takes a step s in C.*

Claim 3.1 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_1$ holds in $C.s$.*

Proof: By the inductive hypothesis, $\mathcal{I}_1$ holds in C. Thus, we have two cases:

1. In C, $(\nexists p : p.current\text{-}status = \dot{w})$.

If in $C.s$, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$, then $\mathcal{I}_1$ holds in $C.s$.

If in $C.s$, $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$, s must be the successful execution of line 14 by p, so $PC_p = 14$ and $US =$ UPGRADING in C. Assume, for the sake of contradiction, that $(\exists q \neq p : PC_q \in \mathrm{CS} \cup \{7\})$ in $C.s$, so $PC_q \in \mathrm{CS} \cup \{7\}$ in C as well. This violates $\mathcal{I}_5$ in C, which contradicts the induction hypothesis.

2. In C, $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$ and $(\forall q \neq p : PC_q \notin \mathrm{CS} \cup \{7\})$.

   If in $C.s$, $(\forall q \neq p : PC_q \notin \mathrm{CS} \cup \{7\})$, then $\mathcal{I}_1$ holds in $C.s$.

   If in $C.s$, $(\exists q \neq p : PC_q \in \mathrm{CS} \cup \{7\})$, then we ree cases for s:

   - s is the completion of Writer-Try() by $\bar{w}$. Thus, in $C.s$, $PC_{\bar{w}} \in \mathcal{L}.\mathrm{CS}_W$. By Corollary 1, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$ in $C.s$, so the claim holds.
   - s is the execution by q of either line 5, finding $US \neq$ UPGRADED, or line 6, finding $\mathit{Permit} =$ TRUE. It follows that, in C, $(\mathit{Permit} = \text{TRUE} \vee US \neq \text{UPGRADED})$. Thus, by $\mathcal{I}_3$, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$ in C. It follows that $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$ in $C.s$, so the claim holds.

In both cases, $\mathcal{I}_1$ holds in $C.s$. □

Claim 3.2 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_2$ holds in $C.s$.*

Proof: Let RZ $= \{2, 3\text{:all}, 4\text{:all}, 5, 6, 7, 8, 10, 11, 12, 13, 14, 15, 20\text{:all}, 21, 22\}$, so $\mathcal{I}_2$ states that $RC = |\{p : PC_p \in \mathrm{RZ}\}|$. By the inductive hypothesis, $\mathcal{I}_2$ holds in C. For all steps s in which RC is unchanged and no process's program counter enters or exits RZ, $\mathcal{I}_2$ clearly holds in $C.s$. For all steps s in which a process's program counter enters RZ (s is the execution of line 1 or line 19), RC is incremented by 1, so $\mathcal{I}_2$ holds in $C.s$. For all steps s in which a process's program counter exits RZ (s is the execution of line 8 or line 22), RC is decremented by 1, so $\mathcal{I}_2$ holds in $C.s$. We have covered all possibilities for s, so $\mathcal{I}_2$ universally holds in $C.s$. □

Claim 3.3.1 *If $\mathcal{I}$ holds in C, then $(\exists p : p.\mathit{current\text{-}status} = \dot{w}) \rightarrow (\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$ holds in $C.s$.*

Proof: By the inductive hypothesis, $\mathcal{I}_3$ holds in configuration C. Thus, we have two cases:

1. In C, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$.

   If in $C.s$, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$, then the claim holds.

   If in $C.s$, $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$, then s must be the successful execution of line 14 by p. It follows that $PC_p = 14$ and $US =$ UPGRADING in C, and $US =$ UPGRADED in $C.s$. By $\mathcal{I}_7$, $\mathit{Permit} =$ FALSE in C, and thus $\mathit{Permit} =$ FALSE in $C.s$. Thus, $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$ in $C.s$, so the claim holds.

2. In C, $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$ and $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$.
   If in $C.s$, $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$, then the claim holds.
   If in $C.s$, $(\mathit{Permit} = \text{TRUE} \vee US \neq \text{UPGRADED})$, then there are four cases for s:
   - $q \neq p$ executes line 2 in s. Suppose, for the sake of contradiction, that q changes US to NULL in $C.s$. It follows that $US = \text{UPGRADING}$ in C, which is a contradiction. Thus, $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$ in $C.s$, so the claim holds.
   - $q \neq p$ executes line 11 or line 13 in s, changing US to UPGRADING or Permit to FALSE, respectively. Thus, $PC_q \in \{11, 13\}$ in C. This violates $\mathcal{I}_1$, which contradicts the induction hypothesis.
   - q executes line 15 in s, so $PC_q = 15$ in C. By $\mathcal{I}_1$ in C, $q \neq p$. Thus, in $C.s$, $(\nexists p : p.\mathit{current\text{-}status} = \dot{w})$, so the claim holds.

In both cases, the claim holds. □

Claim 3.3.2 *If $\mathcal{I}$ holds in C, then $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED}) \rightarrow (\exists p : p.\mathit{current\text{-}status} = \dot{w})$ holds in $C.s$.*

PROOF: By the inductive hypothesis, $\mathcal{I}_3$ holds in C. Thus, we have two cases:

1. In C, $(\mathit{Permit} = \text{TRUE} \vee US \neq \text{UPGRADED})$.
   If in $C.s$, $(\mathit{Permit} = \text{TRUE} \vee US \neq \text{UPGRADED})$, then the claim holds.
   If in $C.s$, $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$, then there are two cases for s:
   - q executes line 13 in s. Thus, $PC_q = 13$ and $US = \text{UPGRADED}$ in C. This violates $\mathcal{I}_6$ in C, which contradicts the induction hypothesis.
   - q executes line 14 successfully in s, changing the value of US from UPGRADING in C to UPGRADED in $C.s$. Therefore, $q.\mathit{current\text{-}status} = \dot{w}$ in $C.s$, so the claim holds.
2. In C, $(\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED})$ and $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$.
   If in $C.s$, $(\exists p : p.\mathit{current\text{-}status} = \dot{w})$, then the claim holds.
   If, in $C.s$, $(\forall p : p.\mathit{current\text{-}status} \neq \dot{w})$, then s must be the execution of line 15 by p. Thus, $\mathit{Permit} = \text{TRUE}$ in $C.s$, so the claim holds.

In both cases, the claim holds. □

Claim 3.3 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_3$ holds in $C.s$.*

PROOF: By Claim 3.3.1 and Claim 3.3.2, we have that if $\mathcal{I}$ holds in C, then $\mathcal{I}_3$ holds in $C.s$. □

Claim 3.4 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_4$ holds in $C.s$.*

PROOF: By the inductive hypothesis, $\mathcal{I}_4$ holds in configuration C. Thus, we have two cases:

1. In C, ($\nexists p : PC_p \in \{11, 12, 13, 14\}$).
   If in $C.s$, ($\nexists p : PC_p \in \{11, 12, 13, 14\}$), then $\mathcal{I}_4$ holds in $C.s$.
   If in $C.s$, ($\exists p : PC_p \in \{11, 12, 13, 14\}$), then s must be the execution of line 10 by p. This step changes the program counter of p only, so $\nexists q \neq p : PC_q \in \{11, 12, 13, 14\}$ in $C.s$.
2. In C, ($\exists p : PC_p \in \{11, 12, 13, 14\}$) and ($\forall q \neq p : PC_q \notin \{11, 12, 13, 14\}$).
   Suppose, for the sake of contradiction, that in $C.s$, ($PC_p \in \{11, 12, 13, 14\}$) and ($\exists q \neq p : PC_q \in \{11, 12, 13, 14\}$). It follows that $PC_q = 10$ and $RC = 1$ in C. However, this violates $\mathcal{I}_2$, contradicting the inductive hypothesis.

In both cases, $\mathcal{I}_4$ holds in $C.s$. □

Claim 3.5 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_5$ holds in $C.s$.*

PROOF: By the inductive hypothesis, $\mathcal{I}_5$ holds in C. Thus, we have two cases:

1. In C, ($\nexists p : PC_p \in \{13, 14\} \vee US \neq \text{UPGRADING}$).
   If in $C.s$, ($\nexists p : PC_p \in \{13, 14\} \vee US \neq \text{UPGRADING}$), then $\mathcal{I}_5$ holds in $C.s$.
   If in $C.s$, ($\exists p : PC_p \in \{13, 14\} \wedge US = \text{UPGRADING}$), then there are two cases for s:
   - p executes line 12 in s. Thus, $RC = 1$ and $US = \text{UPGRADING}$ in C, and also in $C.s$. By Lemma 1, we have $PC_{\bar{w}} \notin \mathcal{L}.\text{CS}_W$ in $C.s$. Additionally, by $\mathcal{I}_2$, we have $\forall q \neq p : PC_q \notin \text{RZ}$ in $C.s$. It follows that ($\forall q \neq p : PC_q \notin \text{CS} \cup \{3\text{:all}, 4\text{:all}, 5, 6, 7\}$) in $C.s$, so the claim holds.
   - $q \neq p$ executes line 11 in s. Thus, $PC_p \in \{13, 14\}$ and $PC_q = 11$ in C. This violates $\mathcal{I}_4$ in C, which contradicts the induction hypothesis.
2. In C, ($\exists p : PC_p \in \{13, 14\} \wedge US = \text{UPGRADING}$) and ($\forall q \neq p : PC_q \notin \text{CS} \cup \{3\text{:all}, 4\text{:all}, 5, 6, 7\}$).
   If in $C.s$, ($\forall q \neq p : PC_q \notin \text{CS} \cup \{3\text{:all}, 4\text{:all}, 5, 6, 7\}$), then $\mathcal{I}_5$ holds in $C.s$.
   If in $C.s$, ($\exists q \neq p : PC_q \in \text{CS} \cup \{3\text{:all}, 4\text{:all}, 5, 6, 7\}$), then there are two cases for s:
   - $q \neq p$ executes line 2 in s. Since $US = \text{UPGRADING}$ in C, $US = \text{NULL}$ in $C.s$. Thus, the claim holds.

- $\bar{w}$ completes $\mathcal{L}$.Writer-Try() in s. Thus, $PC_{\bar{w}} \in \mathcal{L}.\mathrm{CS}_W$ in $C.s$. By Lemma 1, $(\nexists p : PC_p \in \{13, 14\})$ in $C.s$, so the claim holds.

In both cases, $\mathcal{I}_5$ holds in $C.s$. □

Claim 3.6 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_6$ holds in $C.s$.*

Proof: By the inductive hypothesis, $\mathcal{I}_6$ holds in C. Thus, we have two cases:

1. In C, $(\nexists p : PC_p \in \{12, 13, 14\})$.
   If in $C.s$, $(\nexists p : PC_p \in \{12, 13, 14\})$, then $\mathcal{I}_6$ holds in $C.s$.
   If in $C.s$, $(\exists p : PC_p \in \{12, 13, 14\})$, then s must be the execution of line 11 by p. Thus, $US = \textsc{upgrading}$ in $C.s$, so the claim holds.
2. In C, $(\exists p : PC_p \in \{12, 13, 14\})$ and $(US \in \{\textsc{upgrading}, \textsc{null}\})$.
   If in $C.s$, $(US \in \{\textsc{upgrading}, \textsc{null}\})$, then the claim holds.
   If in $C.s$, $(US \notin \{\textsc{upgrading}, \textsc{null}\})$, then s must be the successful execution of line 14 by q. It follows that $PC_q = 14$ in C, so, by $\mathcal{I}_4$, $p = q$. Therefore, $PC_p \notin \{12, 13, 14\}$ in $C.s$. Furthermore, $(\nexists p' : PC_{p'} \in \{12, 13, 14\})$ in $C.s$, so the claim holds.

In both cases, $\mathcal{I}_6$ holds in $C.s$. □

Claim 3.7 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_7$ holds in $C.s$.*

Proof: By the inductive hypothesis, $\mathcal{I}_7$ holds in C. Thus, we have two cases:

1. In C, $(\nexists p : PC_p = 14)$.
   If in $C.s$, $(\nexists p : PC_p = 14)$, then $\mathcal{I}_7$ holds in $C.s$.
   If in $C.s$, $(\exists p : PC_p = 14)$, s must be the execution of line 13 by p. Thus, $Permit = \textsc{false}$ in $C.s$, so the claim holds.
2. In C, $(\nexists p : PC_p = 14)$ and $(Permit = \textsc{false})$.
   If in $C.s$, $(Permit = \textsc{false})$, then $\mathcal{I}_7$ holds in $C.s$.
   If in $C.s$, $(Permit = \textsc{true})$, then s must be the execution of line 15 by $q \neq p$. It follows that $PC_q = 15$ in C, so, by the Upgraded Writer Lemma, $q.\mathit{current\text{-}status} = \dot{w}$ in C. By $\mathcal{I}_1$, $(\nexists p : PC_p = 14)$, so the claim holds.

In both cases, $\mathcal{I}_7$ holds in $C.s$. □

Claim 3.8 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_8$ holds in $C.s$.*

PROOF: By the induction hypothesis, $\mathcal{I}_8$ holds in C. Thus, we have three cases:

1. In C, $(\nexists p : PC_p = 6)$.

   If in $C.s$, $(\nexists p : PC_p = 6)$, then $\mathcal{I}_8$ holds in $C.s$.

   If in $C.s$, $(\exists p : PC_p = 6)$, then s must be the execution of line 5 by p, and $US = \text{UPGRADED}$ in C. We have two cases:

   - $\mathit{Permit} = \text{FALSE}$ in C. Thus, by $\mathcal{I}_3$, $\exists q : q.\mathit{current\text{-}status} = \dot{w}$ in C. Therefore, $(\mathit{Permit} = \text{FALSE})$ and $(q.\mathit{current\text{-}status} = \dot{w})$ in $C.s$, so the claim holds.
   - $\mathit{Permit} = \text{TRUE}$ in C. Thus, $(\mathit{Permit} = \text{TRUE})$ in $C.s$, so the claim holds.

2. In C, $(\exists p : PC_p = 6 \wedge \mathit{Permit} = \text{TRUE})$.

   If in $C.s$, $(\exists p : PC_p = 6 \wedge \mathit{Permit} = \text{TRUE})$, then $\mathcal{I}_8$ holds in $C.s$.

   If in $C.s$, $(\exists p : PC_p = 6 \wedge \mathit{Permit} = \text{FALSE})$, then s must be the execution of line 13 by $q \neq p$. It follows that $PC_q = 13$ in C. This violates $\mathcal{I}_9$ in C, which contradicts the inductive hypothesis.

3. In C, $(\exists p : PC_p = 6 \wedge \mathit{Permit} = \text{FALSE})$ and $(\exists q : q.\mathit{current\text{-}status} = \dot{w})$.

   If in $C.s$, $(\exists q : q.\mathit{current\text{-}status} = \dot{w})$, then $\mathcal{I}_8$ holds in $C.s$.

   If in $C.s$, $(\nexists q : q.\mathit{current\text{-}status} = \dot{w})$, then s must be the execution of line 15 by q. It follows that $\mathit{Permit} = \text{TRUE}$ in $C.s$, so the claim holds.

In all three cases, $\mathcal{I}_8$ holds in $C.s$. $\square$

Claim 3.9 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_9$ holds in $C.s$.*

PROOF: By the inductive hypothesis, $\mathcal{I}_9$ holds in C. Thus, we have two cases:

1. In C, $(\nexists p : PC_p = 6)$.

   If in $C.s$, $(\nexists p : PC_p = 6)$, then $\mathcal{I}_9$ holds in $C.s$.

   If in $C.s$, $(\exists p : PC_p = 6)$, then s must be the execution of line 5 by p, and $US = \text{UPGRADED}$ in C. By $(\forall q : PC_q \notin \{13, 14\})$ in C. It follows that $(\forall q : PC_q \notin \{13, 14\})$ in $C.s$, so the claim holds.

2. In C, $(\exists p : PC_p = 6)$ and $(\forall q : PC_q \notin \{13, 14\}$.

   If in $C.s$, $(\forall q : PC_q \notin \{13, 14\}$, then $\mathcal{I}_9$ holds in $C.s$.

   If in $C.s$, $(\exists q : PC_q \in \{13, 14\})$, then s must be the execution of line 12 by q, and $RC = 1$ in C. By $\mathcal{I}_2$, $(\nexists p : PC_p = 6)$ in C, and in $C.s$ as well. Thus, the claim holds.

In both cases, $\mathcal{I}_9$ holds in $C.s$. $\square$

Claim 3.10.1 *If $\mathcal{I}$ holds in C, then $(PC_{\bar{w}} \in \{19, 20{:}all, 21\}) \rightarrow (DowngradingWriter = \text{TRUE})$ holds in $C.s$.*

Proof: By the induction hypothesis, $\mathcal{I}_{10}$ holds in C. Thus, we have two cases:

1. In C, $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$.
   If in $C.s$, $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$, then the claim holds.
   If in $C.s$, $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\})$, then s must be the execution of line 18 by $\bar{w}$. Thus, $DowngradingWriter = \text{TRUE}$ in $C.s$, so the claim holds.
2. In C, $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\})$ and $(DowngradingWriter = \text{TRUE})$.
   If in $C.s$, $(DowngradingWriter = \text{TRUE})$, then the claim holds.
   If in $C.s$, $(DowngradingWriter = \text{FALSE})$, then s must be the execution of line 21 by $\bar{w}$. It follows that $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$ in $C.s$, so the claim holds.

In both cases, the claim holds. $\square$

Claim 3.10.2 *If $\mathcal{I}$ holds in C, then $(DowngradingWriter = \text{TRUE}) \rightarrow (PC_{\bar{w}} \in \{19, 20{:}all, 21\})$ holds in $C.s$.*

Proof: By the inductive hypothesis, $\mathcal{I}_{10}$ holds in C. Thus, we have two cases:

1. In C, $(DowngradingWriter = \text{FALSE})$.
   If in $C.s$, $(DowngradingWriter = \text{FALSE})$, then the claim holds.
   If in $C.s$, $(DowngradingWriter = \text{TRUE})$, then s must be the successful execution of line 18 by $\bar{w}$. Thus, $PC_{\bar{w}} = 19$ in $C.s$, so the claim holds.
2. In C, $(DowngradingWriter = \text{TRUE})$ and $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\})$.
   If in $C.s$, $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\})$, then the claim holds.
   If in $C.s$, $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$, then s must be the execution of line 21 by $\bar{w}$. Thus, $DowngradingWriter = \text{FALSE}$ in $C.s$, so the claim holds.

In both cases, the claim holds. $\square$

Claim 3.10 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_{10}$ holds in $C.s$.*

Proof: By Claim 3.10.1 and 3.10.2, we have that if $\mathcal{I}$ holds in C, then $\mathcal{I}_{10}$ holds in $C.s$. $\square$

Claim 3.11 *If $\mathcal{I}$ holds in C, then $\mathcal{I}_{11}$ holds in $C.s$.*

Proof: By the induction hypothesis, $\mathcal{I}_{11}$ holds in C. Thus, we have two cases:
$\mathcal{I}_{11}$: $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}) \rightarrow (\forall p \neq \bar{w} : PC_p \notin CS)$.

1. In C, $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$.
   If in $C.s$, $(PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\})$, then the claim holds.
   If in $C.s$, $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\})$, then s must be the execution of line 18 by $\bar{w}$, so $PC_{\bar{w}} = 18$ in C. By Lemma 1, $(\forall p \neq \bar{w} : PC_p \notin CS)$, so we have the claim.
2. In C, $(PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}$ and $(\forall p \neq \bar{w} : PC_p \notin CS)$.
   If in $C.s$, $(\forall p \neq \bar{w} : PC_p \notin CS)$, then the claim holds.
   If in $C.s$, $(\exists p \neq \bar{w} : PC_p \in CS)$, then s must be the execution of line 7 by p and $DowngradingWriter =$ false in C. Thus, by $\mathcal{I}_{10}$, $PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\}$ in C, so $PC_{\bar{w}} \notin \{19, 20\text{:all}, 21\}$ in $C.s$ as well. Therefore, the claim holds.

In both cases, the claim holds. □

We have shown that for each invariant $\mathcal{I}_x$ in $\mathcal{I}$ $(1 \leq x \leq 11)$, if $\mathcal{I}$ holds in C, then $\mathcal{I}_x$ holds in $C.s$, for every possible step s. Therefore, if $\mathcal{I}$ holds in C, then $\mathcal{I}$ holds in $C.s$, for every possible step s. □

5.3 Proof of Theorem 1

Assume that $\mathcal{L}$ satisfies (P1), (P2), (P3), (P4), (P5), and (P6). We will now prove that $\mathcal{L}'$ does as well.

Lemma 4 (Mutual Exclusion) *When a writer is in the CS, no other process can be in the CS.*

Proof: Let p be a writer in the CS. Thus, $p.\textit{current-status} \in \{w, \dot{w}\}$. If $p.\textit{current-status} = \dot{w}$, then, by $\mathcal{I}_1$, there is no other process in the CS. If $p.\textit{current-status} = w$, then $p = \bar{w}$ is the only original writer. By the Writer Lemma and the definition of the CS, $PC_{\bar{w}} \in \mathcal{L}.\mathrm{CS}_W \cup \{21\}$. If $PC_{\bar{w}} \in \mathcal{L}.\mathrm{CS}_W$, then, by Lemma 1, there is no other process in the CS. If $PC_{\bar{w}} = 21$, then, by $\mathcal{I}_{11}$, there is no other process in the CS. □

The following 4 lemmas and corollary will be useful to prove the rest of the properties.

Lemma 5 *Let C be a configuration in which $\exists p : PC_p \in \{4\text{:}all, 5, 6, 7\}$ and $US \neq$ upgraded. Then, for any step s from C, in the resulting configuration $C.s$, if $PC_p \in \{4\text{:}all, 5, 6, 7\}$, then*

$US \neq$ UPGRADED.

PROOF: Suppose, for the sake of contradiction, that in $C.s$, $PC_p \in \{4\text{:all}, 5, 6, 7\}$ and $US =$ UPGRADED. Thus, s must be the successful execution of line 14 by $q \neq p$, so $PC_q = 14$ and $US =$ UPGRADING in C. However, this violates $\mathcal{I}_5$ in C, so we have a contradiction. □

Corollary 5 *Let C be a configuration in which $\exists p : PC_p \in \{4\text{:all}, 5, 6, 7\}$ and $US \neq$ UPGRADED. Let σ be a run from C in which $PC_p \in \{4\text{:all}, 5, 6, 7\}$ in every configuration in σ. Then $US \neq$ UPGRADED in every configuration in σ.*

PROOF: We will prove Corollary 5 by induction on the length of a run σ'' from C so some subsequent configuration C'' in σ consisting of the first k steps in σ.

Base case: $k = 0$. $C'' = C$, so $US \neq$ UPGRADED in C''.

Induction step: $k > 0$. Let σ' be the run from C to some subsequent configuration C' in σ consisting of the first $k-1$ steps of σ. Let s be the k^{th} step of σ such that $C'.s = C''$. By the inductive hypothesis, $US \neq$ UPGRADED in C'. Additionally, by the definition of σ, $PC_p \in \{4\text{:all}, 5, 6, 7\}$ in C' and C''. Therefore, by Lemma 5, $US \neq$ UPGRADED in C''. □

Lemma 6 *Let C be a configuration in which $\exists p : PC_p \in \{4\text{:all}, 5, 6, 7\} \wedge US =$ UPGRADED $\wedge$ $Permit =$ TRUE. Let σ be a run from C in which $PC_p \in \{4\text{:all}, 5, 6, 7\}$ in every configuration in σ. Then $Permit =$ TRUE in every configuration in σ.*

PROOF: We will first prove the following:

Claim 6.1 *Let C be a configuration in which $\exists p : PC_p \in \{4\text{:all}, 5, 6, 7\} \wedge US =$ UPGRADED $\wedge$ $Permit =$ TRUE. Let σ be a run from C in which $PC_p \in \{4\text{:all}, 5, 6, 7\}$ in every configuration in σ. Then in every configuration in σ, $\nexists q : PC_q = 13$.*

PROOF: We will prove Claim 6.1 by induction on the length of a run σ'' from C to some subsequent configuration C'' in σ consisting of the first k steps in σ.

Base case: $k = 0$. $C'' = C$, so $US =$ UPGRADED. By $\mathcal{I}_6$, $\nexists q : PC_q = 13$.

Induction step: $k > 0$. Let σ' be the run from C to some subsequent configuration C' in σ consisting of the first $k-1$ steps of σ. Let s be the k^{th} step of σ such that $C'.s = C''$. By the inductive hypothesis, $\nexists q : PC_q = 13$ in C'. Suppose, for the sake of contradiction, that such a process q exists in C''. Thus, s must be the execution of line 12 by q, and $RC = 1$ in C'. However, this violates $\mathcal{I}_2$, and so we have a contradiction. □

Having proven Claim 6.1, we now prove Lemma 6. Suppose, for the sake of contradiction, that there exists a configuration in σ in which $Permit = \text{FALSE}$. Let C'' be the first configuration in σ in which $Permit = \text{FALSE}$, and let C' be the preceding configuration, with s as the step between them such that $C'.s = C''$. It follows that s is the execution of line 13 by $q \neq p$, so $PC_q = 13$ in C. However, this contradicts Claim 6.1, so we have the lemma. $\square$

Definition 5 A process p is *$\mathcal{L}$-enabled* in a configuration C if p is in the Try section of $\mathcal{L}$ in C and there exists some fixed bound b such that for all runs from C, p enters the CS of $\mathcal{L}$ in at most b of its own steps.

Lemma 7 *Let C be a configuration in which:*

- *r is a read attempt with $PC_r \in \{4\text{:}all, 5, 6, 7\}$,*
- *$\nexists q : q.current\text{-}status = \dot{w}$,*
- *$\bar{w}.current\text{-}status = w \rightarrow PC_{\bar{w}} \notin CS$, and*
- *If $PC_r \in 4\text{:}all$, then r is $\mathcal{L}$-enabled.*

Then r is enabled in C.

PROOF: Suppose, for the sake of contradiction, that r is not enabled in C, i.e. there does not exist a bound b such that in all runs from C, r enters the CS in at most b of its own steps. Therefore, there exists a run σ from C in which $PC_r \in \{4\text{:all}, 5, 6, 7\}$ in every configuration in σ and r executes either line 6 or line 7 more than once.

Suppose that r executes line 7 more than once in σ. Let s be the first step in which r executes line 7 from a configuration C'. It follows that in C', $PC_r = 7$ and $DowngradingWriter = \text{TRUE}$. By $\mathcal{I}_{10}$, $PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}$ in C', so we have a contradiction..

Suppose that r executes line 6 more than once. We have two possible cases:

1. $US \neq \text{UPGRADED}$ in C.

   By Corollary 5, $US \neq \text{UPGRADED}$ in all of σ. Let C' be the first configuration in σ from which r executes line 6. It follows that $PC_r = 6$ and $Permit = \text{FALSE}$ in C'. By $\mathcal{I}_8$, $\exists p$ with $p.current\text{-}status = \dot{w}$ in C'. Thus, by $\mathcal{I}_3$, $US = \text{UPGRADED}$ in C', so we have a contradiction.

2. $US = \text{UPGRADED}$ in C.

   By $\mathcal{I}_3$ applied to C, $Permit \neq \text{FALSE} \vee US \neq \text{UPGRADED}$, so $Permit = \text{TRUE}$. By Lemma 6, $Permit = \text{TRUE}$ in all of σ. Therefore, r executes line 6 at most once in σ, so we have a contradiction.

r executes line 6 and line 7 at most once each in σ, so we have a contradiction. Thus, there exists a bound b such that in all runs from C, r enters the CS in at most b of its own steps. $\square$

Lemma 8 *Let r and r' be any two read attempts in a run σ such that r doorway precedes r'. Then r also $\mathcal{L}$-doorway precedes r', i.e. r completes $\mathcal{L}$.*Reader-Doorway() *before r' begins executing $\mathcal{L}$.*Reader-Doorway() *in σ.*

Proof: Since r completes the doorway before r' begins executing the doorway, there is a configuration in σ in which $PC_r = 4$ and $PC_{r'} = 1$. In this configuration, r has completed $\mathcal{L}$.Reader-Doorway() and r' has not yet began executing $\mathcal{L}$.Reader-Doorway(), so r $\mathcal{L}$-doorway precedes r'. □

Lemma 9 (FIFE among readers) *Let r and r' be any two read attempts in a run σ such that r doorway precedes r'. If r' enters the CS before r, then r is enabled to enter the CS at the time r' enters the CS.*

Proof: Let r and r' be any two read attempts in a run σ such that r doorway precedes r' and r' enters the CS before r. Let C be the earliest configuration in which r' is in the CS, so $r' \in \mathcal{L}.\text{CS}_R$ in C. $\mathcal{L}$ obeys FIFE among readers and, by Lemma 8, r $\mathcal{L}$-doorway precedes r'. Therefore, if $PC_r \in \{4\text{:all}\}$, then r is $\mathcal{L}$-enabled. Additionally, since C is the first configuration in which r' is in the CS, $PC_{r'} \in \{8, 10\}$ and $r'.\textit{current-status} = r$ in C. By $\mathcal{I}_1$, $\nexists p : p.\textit{current-status} = \dot{w}$. By the Writer Lemma, Lemma 1, and $\mathcal{I}_{11}$, $\bar{w}.\textit{current-status} = w \rightarrow PC_{\bar{w}} \notin CS$. Thus, by Lemma 7, r is enabled in C. □

Lemma 10 (Concurrent Entering) *There is an integer b such that, if σ is any run from a reachable configuration such that all writers are in the Remainder section in every configuration in σ, then every read attempt in σ executes at most b steps of the Try section before entering the CS.*

Proof: $\mathcal{L}$ satisfies concurrent entering, so let b_0 be the maximum number of steps that each read attempt executes in $\mathcal{L}$.Reader-Doorway() and $\mathcal{L}$.Reader-WaitingRoom() combined. Suppose, for the sake of contradiction, that in σ, a read attempt r executes more than $b_0 + 5$ steps in the Try section of $\mathcal{L}'$ before entering the CS. Therefore, it must be the case that r executes either line 6 or line 7 more than once in σ.

Suppose r executes line 7 more than once in σ. Let s be the first step in which r executes line 7 from a configuration C in σ. It follows that in C, $PC_r = 7$ and $\textit{DowngradingWriter} = \text{true}$. By $\mathcal{I}_{10}$, $PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}$ in C, so we have a contradiction.

Suppose r executes line 6 more than once in σ. Let C' be the configuration in σ from which r executes line 6. It follows that $PC_r = 6$ and $\textit{Permit} = \text{false}$ in C'. By $\mathcal{I}_8$, $\exists p : p.\textit{current-status} = \dot{w}$ in C', so we have a contradiction

r executes line 6 and line 7 at most once each in σ, so we have a contradiction. Thus, r takes at most $b_0 + 5$ steps in the Try section before entering the CS. □

Lemma 11 (Livelock Freedom) *If no process crashes in an infinite run, then infinitely many attempts complete in that run.*

Proof: Suppose, for the sake of contradiction, that $\mathcal{L}'$ does not satisfy livelock freedom. Let σ be an infinite run in which only finitely many attempts complete in that run. Let C_0 be the configuration in σ following when the last attempt to complete in σ completes, and let σ' be the infinite run from C_0 comprised of the steps following C_0 in σ. Therefore, no attempts complete in σ', and all processes are in the Try or Remainder sections in every configuration in σ'.

$\mathcal{L}$ obeys livelock freedom, so it must be the case that all read attempts in σ' infinitely spin on line 6 or line 7. Suppose that a read attempt r infinitely spins on line 7 in σ'. Thus, there exist infinitely many configurations in σ' in which $\mathit{DowngradingWriter} = \textsc{true}$. By $\mathcal{I}_{10}$, $PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}$ in these configurations, so we have a contradiction.

Suppose that r infinitely spins on line 6 in σ'. Thus, there exist infinitely many configurations in σ' in which $PC_r = 6$ and $\mathit{Permit} = \textsc{false}$. By $\mathcal{I}_8$, $\exists q : q.\mathit{current\text{-}status} = \dot{w}$ in these configurations. By the Upgraded Writer Lemma, $PC_q \in CS$ in these configurations, so we have a contradiction.

It follows that r does not infinitely spin on line 6 or line 7 in σ', so r enters the CS at some point in σ. □

Bounded exit is self-evident. Since there is only one write attempt at a time, $\bar{w}$, FCFS trivially holds.

We now prove that $\mathcal{L}'$ is of the same priority variant as $\mathcal{L}$. The following lemma will be useful in this proof.

Lemma 12 *Let r be a read attempt in a run σ. If in some configuration C in σ, $r >_{rp} \bar{w}$ and $PC_r \in \{4\text{:}all\}$, then $r >_{rp}^{\mathcal{L}} \bar{w}$, i.e. $r >_{rp} \bar{w}$ with respect to $\mathcal{L}$.*

Proof: We have two cases for when $r >_{rp} \bar{w}$:

- r doorway precedes $\bar{w}$. Therefore, there is a configuration in σ in which $PC_r = 4$ and $PC_{\bar{w}} = 16$. It follows that r has completed executing $\mathcal{L}.\textsc{Reader-Doorway}()$ and $\bar{w}$ has not begun executing $\mathcal{L}.\textsc{Writer-Try}()$, so r $\mathcal{L}$-doorway precedes $\bar{w}$. Thus, $r >_{rp}^{\mathcal{L}} \bar{w}$.
- There is a time when some reader or writer p is in the CS, r is in the waiting room, and $\bar{w}$ is in the Try section. $PC_p \in \mathcal{L}.\text{CS}_R$, r is in the waiting room of $\mathcal{L}$, and w is in the Try section of $\mathcal{L}$. Thus, $r >_{rp}^{\mathcal{L}} w$. □

Lemma 13 (Reader-Priority) *If $\mathcal{L}$ satisfies reader-priority, then so does $\mathcal{L}'$: Let r be a read* *attempt in a run σ. If $r >_{rp} \bar{w}$, then $\bar{w}$ does not enter the CS before r.*

Proof: Let r be a read attempt such that $r >_{rp} \bar{w}$ in a run σ, and let C be a configuration in σ in which r is in the waiting room of $\mathcal{L}'$. Suppose $PC_r \in \{5, 6, 7\}$ in C, so $PC_r \in \mathcal{L}.\text{CS}_R$. By Lemma 1, $\bar{w}$ does not enter $\mathcal{L}.\text{CS}_W$ until r has exited $\mathcal{L}.\text{CS}_R$. It follows that $\bar{w}$ does not enter the CS before r in σ.

Suppose $PC_r \in \{4\text{:all}\}$ in C. By Lemma 12, $r >^{\mathcal{L}}_{rp} w$. Since $\mathcal{L}$ satisfies reader-priority, $\bar{w}$ does not enter $\mathcal{L}.\text{CS}_W$ before r enters $\mathcal{L}.\text{CS}_R$. It follows that $\bar{w}$ does not enter the CS before r in σ. □

Lemma 14 (Unstoppable Reader Property) *If $\mathcal{L}$ satisfies the unstoppable reader property, then so does $\mathcal{L}'$: Let C be any reachable configuration in which some read attempt r is in the waiting room.*

- *If, in C, no writer is in the CS or the Exit section and $r >_{rp} \bar{w}$ if $\bar{w}$ is in the Try section, then r is enabled to enter the CS in C.*
- *If a reader r' is in the CS in C, then r is enabled to enter the CS in C.*

Proof: Suppose $\mathcal{L}$ satisfies the unstoppable reader property. We divide the proof into two cases that correspond to the cases of the property:

1. *If, in C, no writer is in the CS or the Exit section and $r >_{rp} \bar{w}$ if $\bar{w}$ is in the Try section, then r is enabled to enter the CS in C.*

   If, in C, $PC_r \in \{4\text{:all}\}$ and $\bar{w}$ is in the Try section, then, by Lemma 12, $r >^{\mathcal{L}}_{rp} \bar{w}$. Since, in C, there is no writer in the CS or Exit section of $\mathcal{L}'$, there is no writer in the CS or Exit section of $\mathcal{L}$. $\mathcal{L}$ satisfies the unstoppable reader property, so r is $\mathcal{L}$-enabled. Thus, by Lemma 7, r is enabled in C.

2. *If a reader r' is in the CS in C, then r is enabled to enter the CS in C.*

   We divide this case into two sub-cases:

   - $\bar{w}$ is not in the Exit section in C.
     By mutual exclusion, there is no writer in the CS in C. Additionally, if $\bar{w}$ is in the Try section in C, then, by definition, $r >_{rp} \bar{w}$. Therefore, this sub-case reduces to case (1) of the unstoppable reader property, for which a proof lies above.
   - $\bar{w}$ is in the Exit section in C.
     It follows that $r'.\mathit{original\text{-}status} = R$, so r' is in $\mathcal{L}.\text{CS}_R$ in C. If, in C, $PC_r \in \{4\text{:all}\}$, then, since $\mathcal{L}$ obeys the unstoppable reader property, r is in the waiting room of $\mathcal{L}$, and a reader is in $\mathcal{L}.\text{CS}_R$ in C, r is $\mathcal{L}$-enabled in C. Additionally, by $\mathcal{I}_1$, $\nexists q : q.\mathit{current\text{-}status} = \dot{w}$. Thus, by Lemma 7, r is enabled in C. □

Lemma 15 (Starvation-Freedom) *If $\mathcal{L}$ satisfies starvation-freedom, then so does $\mathcal{L}'$: If no process crashes in an infinite run, then all attempts complete in that run.*

PROOF: Assume that $\mathcal{L}$ satisfies starvation-freedom. Suppose, for the sake of contradiction, $\mathcal{L}'$ does not satisfy starvation-freedom. Let σ be an infinite run in which no process crashes and not all attempts complete. Since $\mathcal{L}$ satisfies starvation-freedom, it must be the case that some read attempt r infinitely spins on line 6 or line 7 in σ.

Let C' be a configuration in σ in which $PC_r = 6$ if r infinitely spins on line 6 and $PC_r = 7$ if r infinitely spins on line 7. Let P be the set of read attempts not equal to r that are doorway concurrent with r in σ. Consider the infinite run σ' from C' comprised of all steps in σ following C'.

Since $\mathcal{L}$ satisfies livelock freedom, infinitely many attempts complete in σ'. Throughout all of σ', $PC_r = \in \{6, 7\}$, so r is in $\mathcal{L}.\mathrm{CS}_R$. Thus, by Lemma 1, $\bar{w}$ is not in $\mathcal{L}.\mathrm{CS}_W$ in any configuration in σ'. It follows that $\bar{w}$ may complete only one write attempt in σ', since it is possible that $\bar{w}$ is in the middle of an attempt in C'.

Therefore, infinitely many read attempts complete in σ'. After all the read attempts in P that complete in σ' have completed their attempts, the next read attempt r' to enter the CS in σ' is doorway preceded by r. Since $\mathcal{L}'$ satisfies FIFE among readers, r is enabled in the configuration in which r' first enters the CS in σ'. Thus, we have a contradiction. □

We now prove that $\mathcal{L}'$ satisfies the additional desirable properties for upgrade/downgrade support.

Lemma 16 (Upgradeability) *If a reader r in the CS attempts to upgrade and all readers other than r are in the Remainder section, then the upgrade attempt will succeed.*

PROOF: Let C be a configuration in which $PC_r = 10$, and let σ be a run from C to C''', where C''' is the first configuration in σ in which $PC_r \notin \{10, 11, 12, 13, 14\}$. Suppose, for the sake of contradiction, that r's upgrade attempt fails at time t in σ and all other readers are in the Remainder section at t.

Let $s = (C'', r, C''')$ be the last step in σ. There are three cases for s:

- s is the execution of line 10 or line 12 by r, and $RC \neq 1$ in C''. Thus, r's failed upgrade attempt is linearized at $t(s)$. However, this violates $\mathcal{I}_2$ in C'', so we have a contradiction.
- s is the execution of line 14 by r, and $US \neq$ UPGRADING in C''. Let s_0 be the step in σ in which r executed line 11. It follows that in between s_0 and s in σ, some read attempt $r' \neq r$ executed line 2 in step s'. r's failed upgrade attempt is linearized at $t(s')$. This contradicts, since at $t(s')$, there is a reader in the Try section.

Both cases contradict, so r's upgrade attempt does not fail at a time t when all other readers are in the Remainder section. $\square$

Lemma 17 (Non-Upgradeability) *If a reader r in the CS attempts to upgrade and a reader is in the waiting room, then the upgrade attempt will fail.*

PROOF: Suppose, for the sake of contradiction, that a r's upgrade attempt succeeds at time t when a reader r' is in the waiting room, i.e. $PC_{r'} \in \{4\text{:all}, 5, 6, 7\}$. It follows that time $t = t(s)$, where s is a step in which r successfully executes line 14 from a configuration C in which $US = \text{UPGRADING}$. This violates $\mathcal{I}_5$ in C, so we have a contradiction. Therefore, r's upgrade attempt does not succeed at a time t when there is a reader in the waiting room at t. $\square$

Bounded upgrade and bounded downgrade are self-evident.

We now prove upper bounds on the RMR complexity of $\mathcal{L}'$ and the number of shared variables that it uses.

Lemma 18 (RMR Complexity) *If the RMR complexity of $\mathcal{L}$ is $O(s(n))$, where n is the number of attempts, then the RMR complexity of $\mathcal{L}'$ is also $O(s(n))$.*

PROOF: It is clear that the number of RMRs incurred in an attempt A by a process p is bounded by the number of steps that p takes in A. Therefore, not including the procedures of $\mathcal{L}$ invoked in A, p incurs a constant number of RMRs in all procedures except READER-TRY(), which contains lines that can be executed more than once. Suppose p spins on line 6, incurring one RMR when it first reads *Permit* as FALSE from configuration C. By $\mathcal{I}_9$, $\exists q : q.\mathit{current\text{-}status} = \dot{w}$ in C. The next time that *Permit* is updated is when q subsequently downgrades from configuration C'—in which, by $\mathcal{I}_3$, $\mathit{Permit} = \text{FALSE} \wedge US = \text{UPGRADED}$—to configuration C''—in which $\mathit{Permit} = \text{TRUE} \wedge US = \text{UPGRADED}$. By Lemma 6, $\mathit{Permit} = \text{TRUE}$ so long as p remains spinning on line 6. Thus, p incurs exactly one more RMR when it subsequently reads *Permit* as TRUE.

Suppose p spins on line 7, incurring one RMR when it first reads *DowngradingWriter* as TRUE from configuration C. By $\mathcal{I}_{10}$, $PC_{\bar{w}} \in \{19, 20\text{:all}, 21\}$ in C. The next time that *DowngradingWriter* is updated is when $\bar{w}$ subsequently executes line 21. Furthermore, by Lemma 1, *DowngradingWriter* will not be updated by $\bar{w}$ until p has entered and then exited the CS, since $\bar{w}$ cannot exit and reenter the CS before then. Thus, p incurs exactly one more RMR when it subsequently reads *DowngradingWriter* as FALSE.

p executes each procedure of $\mathcal{L}$ at most once. Therefore, the number of RMRs incurred by p is bounded by $O(1) + O(s(n)) = O(s(n))$. $\square$

Lemma 19 (Shared Variables) *If the number of shared variables per process used by $\mathcal{L}$ is*

$O(s'(n))$, where n is the number of attempts, then the number of shared variables per process used by $\mathcal{L}'$ is also $O(s'(n))$. .

Proof: The transformation from $\mathcal{L}$ to $\mathcal{L}'$ adds a constant number of shared variables. Therefore, the number of shared variables used by $\mathcal{L}'$ is bounded by $O(s'(n))$. $\square$

Thus, we have Theorem 1. $\blacksquare$

6 Extension to Multi-Writer Lock

Bhatt and Jayanti proved that a single-writer multi-reader reader-writer lock $\mathcal{L}$ can easily be extended to a multi-writer version by wrapping $\mathcal{L}$ in a mutex lock satisfying starvation freedom, FCFS, and bounded exit [2]. We will not reproduce that work here, since the same method of extension also applies to a single-writer multi-reader reader-writer lock supporting upgrade/downgrade $\mathcal{L}'$. We will only comment that an attempt by an original writer does not release the aforementioned mutex lock until after it has left the Exit section of $\mathcal{L}'$, even if it downgrades. With this care, Bhatt and Jayanti's method of creating a multi-writer lock applies to the transformation in Figure 2.

References

[1] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with “readers” and “writers”. *Commun. ACM*, 14(10):667-668, 1971.

[2] Vibhor Bhatt and Prasad Jayanti. Constant RMR solutions to reader writer synchronization. Technical Report TR2010-662, Dartmouth College, February 2010.

[3] Vibhor Bhatt and Prasad Jayanti. Constant RMR solutions to reader writer synchronization. In *PODC 2010*, pages 468-477, 2010.

[4] Michael Diamond and Prasad Jayanti. Reader-Writer Exclusion Supporting Upgrade and Downgrade with Reader-Priority. Dartmouth College, June 2011.

[5] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569, 1965.

[6] Leslie Lamport. A new solution of Dijkstra’s concurrent programming problem. *Commun. ACM*, 17(8):453-455, 1974.

[7] Vibhor Bhatt. Reader-Writer Lock: Rigorous Formulations and Constant RMR Algorithms. PhD thesis, Dartmouth College, January 2011.