6-2-2011

Reader-Writer Exclusion Supporting Upgrade and Downgrade with Reader-Priority

Michael I. Diamond
Dartmouth College

Dartmouth College Computer Science Technical Report TR2011-683

Reader-Writer Exclusion Supporting Upgrade and Downgrade with Reader-Priority

Michael Diamond

Thesis Advisor: Prasad Jayanti

June 2, 2011

Abstract

The Reader-Writer Exclusion problem [1] seeks to provide a lock that protects some critical section of code for two classes of processes, *readers* and *writers*, where multiple readers are permitted to hold the lock at a time, but only one writer can hold the lock to the exclusion of all other processes. The difficulties in solving this problem lie not only in developing a good algorithm, but in rigorously formulating desirable properties for such an algorithm to have. Recently, Bhatt and Jayanti accomplished both of these tasks for several variants of the Reader-Writer Exclusion problem [2][3]. We seek to extend their work by augmenting one of their algorithms (the one giving readers priority over writers) with the notions of *upgrading* and *downgrading*. We augment the algorithm by allowing processes in the critical section that are only permitted to read to attempt to acquire permission to write by upgrading, and by allowing processes that are permitted to write to relinquish their permission to write–but still remain in the critical section as readers–by downgrading.

Contents

List of Figures

1 Introduction

Reader-Writer Exclusion is a prevalent problem in computer science first formulated by Courtois, Heymans, and Parnas [1]. The problem is an extension of the basic Mutual Exclusion problem formulated by Dijkstra [5]. The idea is that there is some critical section of code that we want to restrict access to in some way. In Mutual Exclusion, we allow only a single process in this critical section at any one time. However, this is not always desirable. There are many scenarios where we want to allow for concurrent read operations, but prevent read operations from being concurrent with write operations and write operations from being concurrent with other write operations. For example, database software might want to allow concurrent transactions that consist only of reading rows, but restrict concurrent transactions where rows are changed. Similarly, file I/O could benefit from the ability to have multiple concurrent readers without the interference of a writer.

While the problem is simple to describe at a high-level, rigorously specifying desirable properties can be more complex. Lamport made considerable advances in this regard when he revisited Dijkstra's Mutual Exclusion problem and introduced the notion of the "doorway" to talk about precedence [6]. More recently, Bhatt and Jayanti rigorously formulated good properties and provided algorithms that satisfy them for several variants of the Reader-Writer Exclusion problem [2][3].

Bhatt and Jayanti's paper provides an asymptotically optimal solution to this problem with a constant number of remote memory references and many other desirable properties. In this paper, we will build on their work by adding functionality to their algorithm. In particular, we will add the ability for processes to change from being a reader to a writer ("upgrading") and vice versa ("downgrading") after they have acquired the lock. We will consider only the variant of Reader-Writer Exclusion where readers have priority over writers. Informally, this means that if any readers want to acquire the lock when a writer doesn't have it, they should succeed, even if a writer has been waiting for it for a long time. We will formulate new properties to define desirable characteristics of upgrading and downgrading, and we will modify some of the existing properties to accommodate these changes. We will then propose an algorithm that satisfies these properties and prove its correctness.

2 The Model

The environment consists of a set of processes with distinct identities (*pid*s) which come from a set $\mathcal{PID}$. Each process has a set of local variables accessible only by that process. In addition, there is a set of shared variables that all processes can access. Our algorithm 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(C, d) behaves as follows: if C's current value is c, C is assigned $c+d$ and c is returned.
- CAS(X, u, v) behaves as follows: if X's current value is u, X is assigned v and *true* is returned; otherwise, X is unchanged and *false* is returned.

Figure 1: Definition of operations F&A and CAS as defined by Bhatt in his paper [4].

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.

A *step* is a triple $(\mathcal{C}, p, \mathcal{C}')$, where $\mathcal{C}$ is a configuration, p is a process, and $\mathcal{C}'$ is the configuration after p executes the instruction at its program counter. We call $\mathcal{C}$ the start configuration, $\mathcal{C}'$ the end configuration, and we say that p has taken a step. A *run from a configuration* $\mathcal{C}_0$ is a list σ of steps such that the start configuration of the first step in σ is $\mathcal{C}_0$, and for each pair of consecutive steps $(\mathcal{C}_a, p, \mathcal{C}_b), (\mathcal{C}_c, p', \mathcal{C}_d) \in \sigma$, $\mathcal{C}_b = \mathcal{C}_c$. A *run* is a run from the initial configuration of the algorithm. A configuration $\mathcal{C}$ is *reachable* if $\mathcal{C}$ is the initial configuration or if there exists a finite run σ such that $\mathcal{C}$ 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 a reachable configuration $\mathcal{C}$ in σ such that p does not take any steps after $\mathcal{C}$.

2.1 Remote Memory References and the Cache-Coherent Model

For reasons detailed by Bhatt and Jayanti [2], our algorithm considers only the cache-coherent (CC) model of memory references. In the CC model, shared variables are stored in global memory separate from any individual process. Each process has a local *cache* wherein it can store copies of shared variables. A distinction is made between *local* and *remote* memory references (RMR). Intuitively, any time a process accesses a local variable or a variable in its cache, that is a local memory reference. When accessing a shared variable, it may or may not be a remote reference depending on whether the operation the process performs modifies the variable or not and whether or not the process has a copy of the shared variable in its cache.

We divide operations into *read* operations (read) and *update* operations (write, fetch&add, compare&swap). We then determine whether a process has a copy of a shared variable in its cache as follows: if a process performs a read operation on a shared variable X at time t and performs another read operation on X at time $t' > t$, then X is considered to be in the process's cache at t' if no process performed an update operation on X at any time t'' such that $t < t'' < t'$. Note that whether the update operation actually changes the value of the shared variable is irrelevant: even a failed compare&swap operation by a process p on a variable X invalidates the cached value of X for every process. We can then formally define a remote memory reference:

Definition 1 A step $(\mathcal{C}, p, \mathcal{C}')$ incurs a *remote memory reference* if in that step, p performed an

update operation on a shared variable or performed a read operation on a shared variable not in its cache.

As a result of this model, a process that is spinning on a shared variable (repeatedly reading it until its value changes to some desirable value) may only experience a small amount of RMR even if it reads the shared variable many times.

2.2 The Readers-Writers Problem with Upgrade and Downgrade

An algorithm for the Readers-Writers problem with Upgrade and Downgrade is broken down into several sections of code: the Remainder section, the Try section, the Critical section (CS), the Upgrade section, the Downgrade section, 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 rights to the CS. In the CS, a process executes sensitive code. While within the CS, a process may execute the Upgrade section (respectively, the Downgrade section) to gain higher privileges (respectively, relinquish some privileges) in the CS. In the Exit section, a process relinquishes its right to the CS. Note that different types of processes might execute different code for the same section.

Each process's code consists of a loop where it executes the Try section, optionally executes the Upgrade and/or Downgrade sections 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 the first line of the Exit section, any line of the Upgrade section, or any line of the Downgrade section. For convenience, when a process enters the CS or finishes executing the Upgrade or Downgrade section, we say its program counter points to the next section of code that it will execute, that is, one of the Exit section, the Upgrade section, or the Downgrade section. 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 (punctuated by an arbitrary number of executions of the Upgrade and Downgrade sections) 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 $t' > t$ that p has completed the Exit section. A is a read attempt if the process is a reader and a write attempt if the process is a writer. A process is said to be *active* at some time t if it started an attempt at some time before t but has not yet finished that attempt at time t.

Each process enters the Try section as a *reader* or *writer* (our algorithm allows these labels to change when a process is in the Remainder section, but we will assume for simplicity that they are fixed). While within the CS, a process can execute the Upgrade section to try to gain the privileges of a writer, or the Downgrade section to release the privileges of a writer. An execution of the Upgrade section may not always be successful: for example, if there are multiple readers present, we can not allow a reader to upgrade its privileges to that of a writer. Therefore, a process can complete the Upgrade section with a result of *success* or *failure*. Downgrade will always be successful, as there is no obstacle to relinquishing privileges. Processes can execute Upgrade and Downgrade an arbitrary number of times while they are within the CS, so to better keep track of

the current status of a given process, we separate processes into the following classes:

Definition 2 At any given time t, all active processes are classified in two different ways:

1. Original Type
   (a) *reader*: a process that executed or is executing the reader's Try section during the current attempt.
   (b) *writer*: a process that executed or is executing the wrtier's Try section during the current attempt.
2. Current Status
   (a) *normal process*: a process that is in the Try section or the Exit section, or that is in the CS and is not executing the Upgrade section or the Downgrade section at time t and that has not completed the Upgrade section with a result of success or the Downgrade section during the current attempt. Alternatively, a process that is not an upgrading process, a downgrading process, an upgraded process, or a downgraded process.
   (b) *upgrading process*: a process that is executing the Upgrade section at time t.
   (c) *downgrading process*: a process that is executing the Downgrade section at time t.
   (d) *upgraded process*: a process that has completed the Upgrade section with a result of success during the current attempt and has not subsequently started executing the Exit section or the Downgrade section.
   (e) *downgraded process*: a process that has completed the Downgrade section during the current attempt and has not subsequently started executing the Exit section, has not subsequently completed the Upgrade section with a result of success, and is not executing the Upgrade section at time t.

A process's classification is considered an additional part of its local state. We then place the following restrictions on the execution of the Upgrade and Downgrade sections:

- A process in the CS may execute the Upgrade section only if it is a normal reader or a downgraded process.
- A process in the CS may execute the Downgrade section only if it is a normal writer or an upgraded process.

When discussing the Try section, it can be useful to break it down further into a *doorway* and a *waiting room* [6]. The doorway is bounded straight-line code where no waiting occurs. The waiting room is all the code that comes after the doorway within the Try section. The concept of a doorway allows us to formulate the notion of one process arriving before another one:

Definition 3 An attempt A *doorway precedes* an attempt A' if the process executing A completes its doorway in A before the process executing A' completes 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 or subsequent attempt of p'.

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

It can also be useful to talk about a process being guaranteed to enter the critical section regardless of the actions of other processes. We formulate the following definition to encapsulate this notion:

Definition 4 A process p is *enabled* to enter the CS in a configuration $\mathcal{C}$ if p is in Try section in $\mathcal{C}$ and there exists some fixed bound b such that for all runs from $\mathcal{C}$, p enters the CS in at most b of its own steps.

2.3 Properties of the Original Algorithm

Listed below are desirable properties for Reader-Writer Exclusion with Reader-Priority as formulated by Bhatt [4]. 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, every process completes the Exit section in at most b of its 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.

  Definition 5 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.

- (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.

- (RP2). Unstoppable Reader Priority : Let $\mathcal{C}$ be any reachable configuration in which some read attempt r is in the waiting room. Then we have:

  1. If a reader is in the CS in $\mathcal{C}$, then r is enabled to enter the CS in $\mathcal{C}$.
  2. If no writer is in the CS or the Exit section in $\mathcal{C}$ and $r >_{rp} w$ holds for all write attempts w that are in the Try section in $\mathcal{C}$, then r is enabled to enter the CS in $\mathcal{C}$.

2.4 Desirable Properties for an Algorithm with Upgrade and Downgrade

Many of the properties originally specified are still of value. However, many of them need to be modified to support the notion of upgrading and downgrading. With this in mind, we formulate the following properties:

- (P1′). Mutual Exclusion : If a normal writer or an upgraded process 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, every process completes the Exit section in at most b of its 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 normal writers are in the Remainder section and there is no upgraded or upgrading process in the CS or Exit 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, in every configuration in σ, all writers are in the Remainder section and there are no upgraded readers or upgrading readers in the CS or the Exit section, 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.

  See Definition 5 above for a definition of $r >_{rp} w$.

- (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.

- (RP2′). Unstoppable Reader Priority : Let $\mathcal{C}$ be any reachable configuration in which some read attempt r is in the waiting room. Then we have:

1. If a normal reader or a downgraded process is in the CS in $\mathcal{C}$, then r is enabled to enter the CS in $\mathcal{C}$.
2. If there is an upgrading process but it is not guaranteed to succeed–that is if there exists a run from $\mathcal{C}$ in which the upgrading process completes its current execution of the Upgrade section with a result of failure–then r is enabled to enter the CS in $\mathcal{C}$.
3. If no writer, upgraded process, or upgrading process is in the CS or the Exit section in $\mathcal{C}$ and $r >_{rp} w$ holds for all write attempts w that are in the Try section in $\mathcal{C}$, then r is enabled to enter the CS in $\mathcal{C}$.

Items 1 and 3 of (RP2′) are analogous to items 1 and 2 of (RP2), but item 2 of (RP2′) requires some explanation. Item 2 of (RP2′) addresses the situation of a reader r' in the CS who has not yet successfully upgraded. In this scenario, we are not necessarily sure whether r' will succeed or not. Since r' is not assured to succeed at its upgrade operation, we don't want r to have to wait for r' if r' completes Upgrade with a result of failure. As a result, we mandate that unless r' is guaranteed to complete Upgrade with a result of success, it must not obstruct r.

Now we must consider desirable properties with regards to upgrading and downgrading. Primarily, we consider when an execution of the Upgrade section should be successful. If another process is in the CS, obviously Upgrade cannot succeed or it would violate mutual exclusion. While Upgrade could conceivably succeed while there are readers in the Try section, this might be undesirable if readers have priority. However, we can safely say that Upgrade should succeed if there are no other readers present. We formulate this property more formally below:

- (P8′). Upgradeability : If a process in the CS executes the Upgrade section and all readers other than that process are in the Remainder section throughout the entire execution of the Upgrade section, the process will complete the Upgrade section with a result of success.
- (P9′). Bounded Upgrade : There is an integer b such that in every run where a process executes the Upgrade section, it completes the Upgrade section in at most b of its steps.
- (P10′). Bounded Downgrade : There is an integer b such that in every run where a process executes the Downgrade section, it completes the Downgrade section in at most b of its steps.

3 A Single-Writer Reader-Priority Algorithm with Upgrade and Downgrade

We begin by proposing a single-writer algorithm which we will later extend to a multi-writer algorithm. The algorithm is a direct extension of Bhatt's reader-priority algorithm [4]. No line of the original algorithm has been modified. Several lines have been added as well as several new procedures. New lines are denoted by decimal notation. E.g. line 16 is a line from the original algorithm while 16.1 is a new line.

Shared Variables
$D \in \{0,1\}$ is a read/write variable, initialized to 0
$Gate \in \{0,1\}$ is a read/write variable initialized to 0
$X \in \mathcal{PID} \cup \{\textit{true}\}$ is a compare&swap variable, initialized to any *pid*
$Permit$ is a Boolean read/write variable, initialized to *true*
C is a fetch&add variable, initialized to 0
$U \in \{\text{CLEAR, UPGRADING, WRITING}\}$ is a compare&swap variable initialized to CLEAR
V is a Boolean read/write variable initialized to *false*

procedure `Writer-Try`$_i$()
REMAINDER SECTION
1. $prevD \leftarrow D$
2. $currD \leftarrow \overline{prevD}$
3. $D \leftarrow currD$
4. $Permit \leftarrow$ *false*
5. `Promote`$_i$()
6. **wait till** $Permit$
CRITICAL SECTION (CS)

procedure `Normal-Writer-Exit`$_i$()
7. $Gate \leftarrow currD$
8. $X \leftarrow i$

procedure `Promote`$_i$()
9. $x \leftarrow X$
10. if $(x \neq$ *true*$)$
11. if (CAS(X, x, i))
12. if $(\neg Permit)$
13. if $(C = 0)$
14. if (CAS$(X, i,$ *true*$)$)
15. $Permit \leftarrow$ *true*

procedure `Reader-Try`$_i$():
REMAINDER SECTION
16. F&A$(C, 1)$
16.1. CAS$(U,$ UPGRADING, CLEAR$)$
17. $d \leftarrow D$
18. $x \leftarrow X$
19. if $(x \in \mathcal{PID})$
20. CAS(X, x, i)
21. if $(X =$ *true*$)$
22. **wait till** $(Gate = d)$
22.1. if $(U =$ WRITING$)$
22.2. **wait till** $V =$ *false*
CRITICAL SECTION (CS)

procedure `Non-Upgraded-Reader-Exit`$_i$():
23. F&A$(C, -1)$
24. `Promote`$_i$()

procedure `Normal-Writer-Downgrade`$_i$():
25.1. F&A$(C, 1)$
25.2. $Gate \leftarrow currD$
25.3 $X \leftarrow i$

procedure `Downgraded-Writer-Exit`$_i$():
26.1. F&A$(C, -1)$

prodedure `Process-Upgrade`$_i$():
27.1. if $(C > 1)$ return *false*
27.2. $U \leftarrow$ UPGRADING
27.3. if $(C > 1)$ return *false*
27.4. $V \leftarrow$ *true*
27.5. return CAS$(U,$ UPGRADING, WRITING$)$

procedure `Upgraded-Process-Downgrade`$_i$()
28.1. $V \leftarrow$ *false*
28.2. $U \leftarrow$ CLEAR

procedure `General-Upgraded-Exit`$_i$()
29.1. `Upgraded-Process-Downgrade`$_i$()
29.2. F&A$(C, -1)$

procedure `Upgraded-Writer-Exit`$_i$()
30.1. `General-Upgraded-Exit`$_i$()

procedure `Upgraded-Reader-Exit`$_i$()
31.1. `General-Upgraded-Exit`$_i$()
31.2. `Promote`$_i$()

Figure 2: Single-Writer Multi-Reader Algorithm satisfying Reader Priority with Upgrade and Downgrade. Code for process with pid i.

3.1 Restrictions on Procedures

The above algorithm utilizes many different procedures for the various types of processes and it may not be immediately clear which processes can execute which procedures at what times. Figure 3 summarizes this information, including valid locations for the program counter (PC) of a process at a given time. Note that since procedure calls only affect local state, being on a line with a procedure call is the same as being on the first line of that procedure, e.g. PC $= 31.1 \implies$ PC $= 29.1 \implies$ PC $= 28.1$.

Valid Procedures and PC Values for a Process in the CS

Original Type	Current Status	Valid Procedure	Valid PC Location
reader	normal	`Process-Upgrade` `Non-Upgraded-Reader-Exit`	27.1 23
reader	downgraded	`Process-Upgrade` `Non-Upgraded-Reader-Exit`	27.1 23
reader	upgraded	`Upgraded-Process-Downgrade` `Upgraded-Reader-Exit`	28.1 $31.1 \Rightarrow 29.1 \Rightarrow 28.1$
writer	normal	`Normal-Writer-Downgrade` `Normal-Writer-Exit`	25.1 7
writer	downgraded	`Process-Upgrade` `Downgraded-Writer-Exit`	27.1 26.1
writer	upgraded	`Upgraded-Process-Downgrade` `Upgraded-Writer-Exit`	28.1 $30.1 \Rightarrow 29.1 \Rightarrow 28.1$

Figure 3: Mapping of process types to procedures and PC values.

3.2 Informal Description of the Algorithm

First we will offer a brief description of the original algorithm. For this portion, you can ignore the lines that have decimal notation (e.g. 16.1). For a more detailed description, see Bhatt's paper [4].

First, a short description of the shared variables:

- C: The C variable is a count of all readers currently in the system.
- D: The D variable indicates a direction. The idea is that when the writer is in the CS, readers will wait to approach the CS from a particular direction (read from the D variable). When the writer leaves the CS, it will let all readers waiting to come from this direction into the CS.
- $Gate$: The $Gate$ variable indicates which direction is allowed into the CS. Readers that have a local d variable that matches $Gate$ are allowed into the CS.
- X: The X variable indicates whether the writer owns the CS or not. If X is *true*, no reader should be in the CS (except when the writer is exiting). Processes write their own pids into X to disrupt other processes that might be trying to set X to *true*.

- $Permit$: The $Permit$ variable indicates whether the writer is trying to gain permission to enter the CS. When *false*, the writer is trying to gain permission to enter the CS. When *true*, if the writer is in the waiting room, it has permission to enter the CS. Otherwise, the writer isn't trying to gain access to the CS.

When a reader enters (Lines 16 through 22), it announces its presence by updating the C variable. The reader then reads the D variable into d. Then the reader reads the X variable, and if it's not *true* (i.e. the writer doesn't own the CS), it tries to CAS it's own pid into X to prevent anyone from setting X to *true* (giving ownership of the CS to the writer). Then it checks to see if the writer owns the CS. If so, it waits until the writer lets it in from the direction d it read earlier. When a reader exits (Lines 23 through 24), it updates the C variable appropriately and then tries to promote the writer (if present) into the CS. The mechanics of `Promote` aren't relevant to Upgrade and Downgrade, so their explanation will be omitted here.

When the writer enters (Lines 1 through 6), it first changes the direction readers should approach the CS from when the writer is in the CS. Then the writer indicates that it wants to gain access to the CS by setting $Permit$ to *false*. It then tries to promote itself into the CS. Finally, it waits until $Permit$ is *true*, meaning it is allowed to enter the CS. When the writer exits (Lines 7 through 8), it lets any readers waiting to enter the CS in by setting $Gate$ to match the direction D and clears X to indicate that it no longer owns the CS.

The addition of Upgrade and Downgrade largely ignores the above algorithm. The algorithm as presented above is designed to handle synchronization between readers and a single writer. However, when the writer downgrades, it effectively turns itself into a reader. As a result, the portion of the algorithm concerned with synchronization between readers and the writer can be largely ignored when handling upgrading and downgrading.

With this in mind, we will walk through the relevant aspects of the Upgrade and Downgrade system. First, we will discuss two new shared variables that are introduced:

- U: The U variable indicates upgrade status. If it is CLEAR, no one is upgraded and no one is currently trying to upgrade (or if someone is trying to upgrade, they will fail). If it is UPGRADING, someone is currently trying to upgrade and might succeed. If it is WRITING, someone has successfully upgraded and currently has permission to write in the CS.

- V: The V variable is a variable for processes to spin on when $U =$ WRITING. If it is *true* when $U =$ WRITING, it will keep readers from entering the CS. This separate spin variable is necessary to avoid cache misses that would occur if processes spun on U.

When a reader enters (Lines 16 through 22.2), it announces its presence by updating the C variable. Remember that we only need Upgrade to succeed if there are no readers in the system. If C gets above 1, we are allowed to make any upgrading process fail. After updating C, the reader tries to thwart a process that might be currently upgrading by clearing the upgrade status (U). Next it proceeds through the steps necessary to avoid conflict with writers (described above). Then it checks if a process has successfully upgraded by checking the U variable. If U is WRITING (i.e. an upgraded process is in the CS), it will wait on the V variable. When V is *false*, meaning that the upgraded process has left the CS (or downgraded), it will be able to enter the CS. If U is not WRITING, it can safely enter the CS, since any upgrade effort in progress is bound to fail.

Why is this so? Let's examine the Upgrade section (Lines 27.1 through 27.5) to find out. Let's say there's some process u executing the Upgrade section. The Upgrade section is designed to fail if possible. There are three points that it can fail at: Line 27.1, 27.3 or 27.5. If there are any other readers (or downgraded writers) in the system (that is, in the Try section or CS) when u executes Line 27.1 or Line 27.3, u will immediately fail because $C > 1$. So if there is a process on Line 22.1 and it finds $U \neq$ WRITING, it can safely proceed into the CS without worry that a process on Line 27.1, 27.2, or 27.3 will be able to succeed at Upgrade. What if u is at Line 27.4 or 27.5, though? Well, for u to succeed, U must be UPGRADING when u performs the CAS on Line 27.5. And for u to be at Line 27.4 or 27.5, it must have read $C = 1$ at some point. But at some time after that, another process came and got to Line 22.1. To do that, this new process would have had to execute Line 16 (updating C to a value > 1) and then 16.1, clearing the U variable. Additionally, no new process could enter Upgrade to restore U to UPGRADING, because Line 27.1 prevents two processes from being in the Upgrade section at the same time. Therefore, if there's a process at Line 22.1 before u executes Line 27.5, u will fail the CAS on 27.5.

In contrast to the Upgrade section, the Exit section and Downgrade section for upgraded processes are straightforward. When an upgraded process is downgrading (Lines 28.1 through 28.2), all it needs to do is set V to *false* to release readers stuck at Line 22.2 and then set U to CLEAR to indicate no upgraded process is in the CS. When an upgraded process is exiting (Lines 29.1 through 29.2), it does the same thing, but then also decrements C to indicate that it's leaving the system. An exiting reader would also try to promote a writer into the CS if applicable (Line 31.2). The Downgrade section for a normal writer (Lines 25.1 through 25.3) is also straightforward. The writer first adds itself to the system as a reader by incrementing C and then performs a normal writer exit, allowing any waiting readers to join it in the CS. When a downgraded writer exits, all it needs to do is decrement C.

We will now address the need for the V variable. Without the V variable, processes would just spin on U. The problem comes when new readers enter the Try section. If processes just spin on U, then we can construct the following bad interleaving: a process is upgraded ($U =$ WRITING) and a process p is spinning on U waiting for it to change to a value other than WRITING. Now a process enters and executes line 16.1. The CAS fails since $U \neq$ UPGRADING, but because it is an update operation, all cached copies of U are invalidated. Now p suffers a cache miss the next time it reads U. Every reader that enters could cause a cache miss for p leading to non-constant RMR.

4 Proof of Correctness

Theorem 1 (Single-Writer Multi-Reader lock with Reader Priority and Upgrade/Downgrade) *The algorithm in Figure 2 implements a Single-Writer Multi-Reader lock satisfying properties (P1′), (P2)-(P4), (P5′), (P6), (RP1) and (RP2′), (P8′)-(P10′). The RMR complexity of the algorithm in the CC model is $O(1)$. The algorithm uses $O(1)$ number of shared variables that support read, write, fetch&add, and compare&swap operations.*

Note that the form of the proof and methods (proof by invariants) used are directly adapted from the original proof by Bhatt and Jayanti since the entire algorithm must be proved correct again in addition to the new components. As a result, much of the following language is taken directly from Bhatt's original paper with modifications to allow different types of invariants [4].

In particular, the following invariants and their proofs are taken verbatim from the paper with no substantive modification:
$\mathcal{I}_{1,2,3}$, $\mathcal{I}_4$, $\mathcal{I}_{6tf}$, $\mathcal{I}_{6tt}$

The following invariants and their proofs are taken almost verbatim from Bhatt and Jayanti's paper with minor substantive modifications:
$\mathcal{I}_9$, $\mathcal{I}_{10,11}$, $\mathcal{I}_{12...14}$, $\mathcal{I}_{ptf}$, $\mathcal{I}_{ptt}$, $\mathcal{I}_{15}$, $\mathcal{I}_6$, $\mathcal{I}_7$, $\mathcal{I}_8$

The remaining invariants and their proofs are original and serve to prove the correctness of the additions to the algorithm.

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

- $\mathcal{I}$ holds in the initial configuration, and
- If $\mathcal{I}$ holds in configuration $\mathcal{C}$, then it also holds in any configuration $\mathcal{C}.s$ resulting from a step s performed by any process. Note that some steps s might be impossible because $\mathcal{I}$ holds in $\mathcal{C}$.

4.1 Invariants of the Algorithm

As defined earlier, configuration $\mathcal{C}$ corresponds to the value of shared variables D, $Gate$, X, $Permit$, C, U, and V and the local state of each individual process. The state of the reader r is given by its program counter, PC_r, the values of its local variables d and x, and whether it is normal, upgraded, or downgraded. The state of the writer w is given by its program counter, the value of its local variable x, and whether it is normal, upgraded or downgraded.

4.1.1 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 process p by $p.y$. Similarly, we denote the pid of process p by $p.pid$.
- PC_p is the program counter of a process p. Since there's only one writer, we call it w. Note that at any time t, $\text{PC}_w \in \{1 \ldots 15, 25.1 \ldots 29.2\}$, and for a reader r, $\text{PC}_r \in \{16 \ldots 24, 9 \ldots 15, 27.1 \ldots 29.2\}$. As the statements at Line 5, Line 24, and Line 31.2 are calls to the `Promote` procedure, we will assume $\text{PC} \in \{5, 24, 31.2\} \implies \text{PC} = 9$. Similarly, since the statements at Line 31.1 and Line 30.1 are calls to the `General-Upgraded-Exit` procedure, we will assume $\text{PC} \in \{31.1, 30.1\} \implies \text{PC} = 29.1$. And finally, since the statement at Line 29.1 is a call to the `Upgraded-Process-Downgrade` procedure, we will assume $\text{PC} = 29.1 \implies \text{PC} = 28.1$. Note that this means that $\text{PC} \in \{31.1, 30.1\} \implies \text{PC} = 28.1$. These assumptions are valid since procedure calls only modify the local state of the process. Therefore, it is not necessary to consider any situations where $\text{PC} \in \{5, 24, 31.2, 31.1, 30.1, 29.1\}$.

- $\mathcal{R}(\textit{condition A})$ denotes the set of readers which satisfy the *condition A*, e.g., $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \ldots 22\})$ is the set of all readers r, such that $\text{PC}_r \in \{18 \ldots 22\}$ and the value of the local variable $r.d$ is equal to $\overline{D}$. For the purposes of this notation, no distinction is made between the various types of readers, so for example, if an upgraded reader is in the critical section, $|\mathcal{R}\{\text{PC} = 28.1\}| = 1$.

- Similarly, $\mathcal{W}(\textit{condition A})$ denotes the set of writers which satisfy the *condition A*. Since there is only one writer, this set either contains a single element or is empty. For the purposes of this notation, no distinction is made between the various types of writers.

- Similarly, $\mathcal{P}(\textit{condition A})$ denotes the set of processes (readers or writers) which satisfy the *condition A*. For the purposes of this notation, no distinction is made between the various types of processes.

- $\mathcal{PID}(H)$, denotes the set of pids corresponding to the processes with their program counters coming from set H. More formally, $\mathcal{PID}(H) = \{p.pid : \text{PC}_p \in H\}$.

Now we describe the set of invariants $\mathcal{I}$ satisfied by any configuration $\mathcal{C}$ of the algorithm. The set $\mathcal{I}$ comprises of several different invariants listed in Figure 4.

We have classified much of $\mathcal{I}$ by the program counter of the writer, i.e., PC_w. For example, $\mathcal{I}_{1,2,3}$ is the invariant when $\text{PC}_w \in \{1 \ldots 3\}$. Of the invariants conditioned on PC_w, only one ever applies at a time. These invariants are: $\mathcal{I}_{1,2,3}$, $\mathcal{I}_4$, $\mathcal{I}_9$, $\mathcal{I}_{10,11}$, $\mathcal{I}_{12 \ldots 14}$, $\mathcal{I}_{ptf}$, $\mathcal{I}_{ptt}$, $\mathcal{I}_{15}$, $\mathcal{I}_6$, $\mathcal{I}_{6tf}$, $\mathcal{I}_{6tt}$, $\mathcal{I}_7$, $\mathcal{I}_8$, $\mathcal{I}_{25.1/2}$, $\mathcal{I}_{25.3}$, $\mathcal{I}_{down}$.

There are a couple global invariants, i.e. invariants which are unconditionally true. For instance, $\mathcal{I}_G$ applies regardless of the PCs of the processes.

There are also some invariants conditioned on the number of processes or readers at a given line. For instance, $\mathcal{I}_{clear}$ applies when $|\mathcal{P}(\text{PC} = 28.2)| = 1$, that is, when there is exactly one process at line 28.2.

And finally, there are some invariants conditioned on the state of the shared variables. For instance, $\mathcal{I}_{uw}$ depends on the value of the U variable.

- $\mathcal{I}_G$: Global Invariant

  1. $C = |\mathcal{R}(\text{PC} \in \{16.1 \dots 23\}) \cup \mathcal{W}(\text{PC} \in \{25.2 \dots 26.1\}) \cup \mathcal{P}(\text{PC} \in \{27.1 \dots 29.2\})|$

- $\mathcal{I}_{1,2,3}$: $\text{PC}_w = \{1, 2, 3\} \implies$

  1. $Gate = D$
  2. $X \neq$ *true*, $X \notin \mathcal{PID}(13, 14)$
  3. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 23, 27.1 \dots 29.2\}) = \mathcal{R}(\text{PC} = 15) = \emptyset$
  4. $Permit =$ *true*

- $\mathcal{I}_4$: $\text{PC}_w = 4 \implies$

  1. $Gate = \overline{D}$
  2. $X \neq$ *true*, $X \notin \mathcal{PID}(13, 14)$
  3. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  4. $Permit =$ *true*

- $\mathcal{I}_9$: $\text{PC}_w = 9 \wedge X \neq$ *true* $\implies$

  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *false*
  4. $X \in \mathcal{PID}(14) \implies \mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \subseteq \mathcal{R}(x = X)$

- $\mathcal{I}_{10,11}$: $\text{PC}_w \in \{10, 11\} \wedge X \neq$ *true* $\implies$

  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *false*
  4. $X \in \mathcal{PID}(14) \implies \mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \subseteq \mathcal{R}(x = X)$
  5. $X \neq w.x \implies \mathcal{R}(\text{PC} \in \{16.1 \dots 24, 27.1 \dots 29.2, 9\}) \cup \mathcal{R}(x = X, \text{PC} \in \{10, 11\}) \neq \emptyset \vee X \in \mathcal{PID}(12 \dots 14))$

- $\mathcal{I}_{12 \dots 14}$: $\text{PC}_w \in \{12 \dots 14\} \wedge X \neq$ *true* $\implies$

  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *false*
  4. $X \in \mathcal{PID}(14) \implies \mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \subseteq \mathcal{R}(x = X)$
  5. $X \neq w.pid \implies \mathcal{R}(\text{PC} \in \{16.1 \dots 24, 27.1 \dots 29.2, 9\}) \cup \mathcal{R}(x = X, \text{PC} \in \{10, 11\}) \neq \emptyset \vee X \in \mathcal{PID}(12 \dots 14))$

- $\mathcal{I}_{ptf}$: $\text{PC}_w \in \{9 \dots 14\}$, $X =$ *true* and $Permit =$ *false* $\implies$

  1. $Gate = \overline{D}$
  2. $|\mathcal{R}(\text{PC} = 15)| = 1$
  3. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset$

- $\mathcal{I}_{ptt}$: $\text{PC}_w \in \{9 \dots 14\}$, $X =$ *true* and $Permit =$ *true* $\implies$

  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset$

- $\mathcal{I}_{15}$: $\text{PC}_w = 15 \implies$

  1. $Gate = \overline{D}$
  2. $r(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *false*
  4. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset$
  5. $X =$ *true*

- $\mathcal{I}_6$: $\text{PC}_w = 6$ and $X \neq$ *true* $\implies$

  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *false*
  4. $X \in \mathcal{PID}(14) \implies \mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \subseteq \mathcal{R}(x = X)$
  5. $\mathcal{R}(\text{PC} \in \{16.1 \dots 24, 27.1 \dots 29.2, 9\}) \cup \mathcal{R}(x = X, \text{PC} \in \{10, 11\}) \neq \emptyset \vee X \in \mathcal{PID}(12 \dots 14)$

- $\mathcal{I}_{6tf}$: $\text{PC}_w = 6$, $X =$ *true*, $Permit =$ *false* $\implies$ Identical to $\mathcal{I}_{ptf}$.
- $\mathcal{I}_{6tt}$: $\text{PC}_w = 6$, $X =$ *true*, $Permit =$ *true* $\implies$ Identical to $\mathcal{I}_{ptt}$.
- $\mathcal{I}_7$: $\text{PC}_w = 7 \implies$
  1. $Gate = \overline{D}$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *true*
  4. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \ldots 23, 27.1 \ldots 29.2\}) = \mathcal{R}(\text{PC} \in \{22.1 \ldots 23, 27.1 \ldots 29.2\}) = \emptyset$
  5. $X =$ *true*
- $\mathcal{I}_8$: $\text{PC}_w = 8 \implies$
  1. $Gate = D$
  2. $\mathcal{R}(\text{PC} = 15) = \emptyset$
  3. $Permit =$ *true*
  4. $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \ldots 23, 27.1 \ldots 29.2\}) = \emptyset$
  5. $X =$ *true*
- $\mathcal{I}_{25.1/2}$: $\text{PC}_w \in \{25.1 \ldots 25.2\} \implies$ Identical to $\mathcal{I}_7$.
- $\mathcal{I}_{25.3}$: $\text{PC}_w = 25.3 \implies$ Identical to $\mathcal{I}_8$ with the additional item:
  6. $\mathcal{R}(\text{PC} \in \{27.2 \ldots 29.2\}) = \emptyset$
- $\mathcal{I}_{down}$: $\text{PC}_w \in \{26.1 \ldots 29.2\} \implies$ Identical to $\mathcal{I}_{1,2,3}$.
- $\mathcal{I}_{G2}$: $|\mathcal{P}(\text{PC} \in \{27.2 \ldots 29.2\})| \leq 1$
- $\mathcal{I}_{uw}$: $U = \text{WRITING} \implies |\mathcal{P}(\text{PC} \in \{28.1 \ldots 28.2\})| = 1$
- $\mathcal{I}_{clear}$: $|\mathcal{P}(PC = 28.2)| = 1 \implies V =$ *false*
- $\mathcal{I}_{stuck}$: $\mathcal{R}(PC = 22.2) \neq \emptyset \implies \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) = \emptyset$
- $\mathcal{I}_{vt}$: $V =$ *true* $\wedge\, \mathcal{R}(\text{PC} = 22.2) \neq \emptyset \implies |\mathcal{P}(\text{PC} = 28.1)| = 1$
- $\mathcal{I}_{block}$: $|\mathcal{P}(\text{PC} \in \{27.5, 28.1\})| = 1 \implies V =$ *true*
- $\mathcal{I}_{wri}$: $|\mathcal{P}(PC = 28.1)| = 1 \implies U = \text{WRITING}$
- $\mathcal{I}_{uu}$: $U = \text{UPGRADING} \wedge |\mathcal{P}(PC \in \{27.4, 27.5\})| = 1 \implies \mathcal{R}(\text{PC} \in \{17 \ldots 22.2\}) = \emptyset$
- $\mathcal{I}_{read}$: $\mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) \neq \emptyset \implies U = \text{CLEAR} \vee \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) = \emptyset$
- $\mathcal{I}_{uwvt}$: $U = \text{WRITING} \wedge V =$ *true* $\implies \mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) = \emptyset$

Figure 4: Invariants comprising $\mathcal{I}$.

4.2 Proof of the Invariants

Now we prove the correctness of the set of invariants $\mathcal{I}$ given above. We will first show that $\mathcal{I}$ holds initially.

Lemma 2 *If $\mathcal{C}_0$ is the initial configuration of the algorithm given in Figure 2, then $\mathcal{I}$ holds in $\mathcal{C}_0$.*

PROOF. Initially in the algorithm: $\text{PC}_w = 1$ and for all readers r, $\text{PC}_r = 16$; $C = 0$; $U = \text{CLEAR}$; and $V =$ *false*. Therefore $\mathcal{I}_G, \mathcal{I}_{G2}, \mathcal{I}_{uw}, \mathcal{I}_{clear}, \mathcal{I}_{vt}, \mathcal{I}_{block}, \mathcal{I}_{wri}, \mathcal{I}_{uu}, \mathcal{I}_{read}, \mathcal{I}_{uwvt}, \mathcal{I}_{stuck}$ all trivially hold.

$\mathcal{I}_{1,2,3}$ is the only other invariant which should be apply in $\mathcal{C}_0$. Items 1,2 and 4 of $\mathcal{I}_{1,2,3}$ are true in $\mathcal{C}_0$ because initially $Gate = D, X \neq$ *true* and $Permit =$ *true*. Item 3 of $\mathcal{I}_{1,2,3}$ is true because for all readers r, $\text{PC}_r = 16$. □

Now we will prove the main lemma required to prove the correctness of the invariants. Essentially, we have to show that if $\mathcal{I}$ holds in a configuration, then it also holds in a configuration resulting from a step by any process.

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

PROOF.

First we will show that our global invariants hold for any step by any process. Note that many of the following invariant proofs rely on certain processes not being permitted to execute certain procedures based on their state as described in Figure 3. For instance, it is impossible for a process on Line 28.2 to step to Line 28.1: a downgraded process is prohibited from downgrading again until it upgrades.

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

PROOF. We have to show that $C = |\mathcal{R}(\text{PC} \in \{16.1 \ldots 23\}) \cup \mathcal{W}(\text{PC} \in \{25.2 \ldots 26.1\}) \cup \mathcal{P}(\text{PC} \in \{27.1 \ldots 29.2\})|$. Let us first define $\mathcal{R}_G$ to be the set of readers with PC pointing to one of the lines specified in this invariant and let $\mathcal{W}_G$ be the set of writers with PC pointing to one of the lines specified in this invariant. We then have:

$$\mathcal{R}_G = \mathcal{R}(\text{PC} \in \{16.1 \ldots 23\}) \cup \mathcal{P}(\text{PC} \in \{27.1 \ldots 29.2\})$$
$$\mathcal{W}_G = \mathcal{W}(\text{PC} \in \{25.2 \ldots 26.1\}) \cup \mathcal{P}(\text{PC} \in \{27.1 \ldots 29.2\})$$

Note that C only increases (and only by 1) when a reader r takes a step at Line 16 (i.e., r joins the set $\mathcal{R}_G$) or the writer w takes a step at Line 25.1 (i.e. w joins the set $\mathcal{W}_G$).

Similarly, C only decreases (and only by 1) when a reader r takes a step at Line 23 or 29.2 (i.e., r leaves the set $\mathcal{R}_G$) or the writer takes a step at Line 26.1 or 29.2 (i.e. w leaves the set $\mathcal{W}_G$). That the above-mentioned lines (16, 25.1, 23, 29.2, 26.1) are the only entry and exit points for $\mathcal{R}_G$ and $\mathcal{W}_G$ is obvious from the algorithm and the rules for executing procedures formulated in Figure 3. Therefore $\mathcal{I}_G$ holds for any step s. □

Claim 3.2 *If $\mathcal{I}_{G2}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. First note that the set of PC values specified in the invariant ($\{27.2 \ldots 29.2\}$) can only be entered from one line (Line 27.2) because of the rules for executing procedures and because $\mathcal{I}_{G2}$ holds in $\mathcal{C}$. Therefore, to violate $\mathcal{I}_{G2}$, s must be a step from Line 27.1 to Line 27.2 when there is already a process with PC $\in \{27.2 \ldots 29.2\}$ in $\mathcal{C}$. By $\mathcal{I}_G$, $C > 1$ in configuration $\mathcal{C}$. Therefore, s cannot possibly be a step from Line 27.1 to Line 27.2. □

Claim 3.3 *If $\mathcal{I}_{uw}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $U =$ WRITING in $\mathcal{C}$. To violate $\mathcal{I}_{uw}$, s must be a step from Line 27.5 to Line 28.1 or s must be a step from Line 28.2 to 29.2, 26.1, or 23. If $U =$ WRITING, the CAS at line 27.5 would fail, so the first case is impossible. And if s is a step from Line 28.2, U will be set to CLEAR.

- Assume $U \neq$ WRITING $\wedge\ |\mathcal{P}(\text{PC} \in \{28.1 \ldots 28.2\})| \neq 1$ in $\mathcal{C}$. Then to violate $\mathcal{I}_{uw}$, s must set U to WRITING. However, the only place U can be set to WRITING is on Line 27.5 by a successful Upgrade, so s would have to be a step from Line 27.5 to 28.1 (see Figure 3). By $\mathcal{I}_{G2}$, there can be no process on Line 28.1 or 28.2 in $\mathcal{C}$, so $|\mathcal{R}(\text{PC} \in \{28.1 \ldots 28.2\})| = 1$ after s.

□

Claim 3.4 *If $\mathcal{I}_{clear}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $|\mathcal{P}(PC = 28.2)| = 1$ in $\mathcal{C}$. V can only be set to *true* on Line 27.4. By $\mathcal{I}_{G2}$, there can be no process on Line 27.4 in $\mathcal{C}$.
- Assume $|\mathcal{P}(PC = 28.2)| \neq 1 \wedge V =$ *true* in $\mathcal{C}$. By $\mathcal{I}_{G2}$, $|\mathcal{P}(PC = 28.2)| = 0$ in $\mathcal{C}$. Therefore, to violate $\mathcal{I}_{clear}$, a process would have to step from Line 28.1 to 28.2 without setting V to *false* which is impossible.

□

Claim 3.5 *If $\mathcal{I}_{stuck}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $\mathcal{R}(PC = 22.2) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{stuck}$, s must be a step from Line 27.3 to Line 27.4. By $\mathcal{I}_G$, $C > 1$, so this step is impossible.
- Assume $\mathcal{R}(PC = 22.2) = \emptyset \wedge \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{stuck}$, s must be a step from Line 22.1 to Line 22.2, so U must be equal to WRITING. By $\mathcal{I}_{uw}$, there is a process at Line 28.1 or 28.2 in $\mathcal{C}$. However, this violates $\mathcal{I}_{G2}$. Therefore, s is impossible.

□

Claim 3.6 *If $\mathcal{I}_{vt}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $V =$ *true* $\wedge\ \mathcal{R}(\text{PC} = 22.2) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{vt}$, either another process would have to step to Line 28.1 or the process at Line 28.1 would have to step Line 28.2 without setting V to *false*. The first case is impossible by $\mathcal{I}_{G2}$ and the second is impossible since V is set to *false* on Line 28.1.
- Assume $V =$ *false* $\wedge\ \mathcal{R}(\text{PC} = 22.2) \neq \emptyset \wedge |\mathcal{P}(\text{PC} = 28.1)| \neq 1$ in $\mathcal{C}$. To violate $\mathcal{I}_{vt}$, s must set V to *true* which can only occur on Line 27.4. By $\mathcal{I}_{stuck}$, there is no process on Line 27.4 in $\mathcal{C}$.

- Assume $V = \textit{true} \wedge \mathcal{R}(\text{PC} = 22.2) = \emptyset \wedge |\mathcal{P}(\text{PC} = 28.1)| \neq 1$ in $\mathcal{C}$. To violate $\mathcal{I}_{vt}$, a process would have to step from Line 22.1 to Line 22.2. For a process to step from Line 22.1 to Line 22.2, WRITING must be *true* in $\mathcal{C}$. Therefore, by $\mathcal{I}_{uw}$, there must be a process on Line 28.1 or 28.2 in $\mathcal{C}$. Since there can be no process on Line 28.1 (by the assumption $|\mathcal{P}(\text{PC} = 28.1)| \neq 1$ and $\mathcal{I}_{G2}$), there must be a process on Line 28.2 in $\mathcal{C}$. But by $\mathcal{I}_{clear}$, $V =$ *false* in $\mathcal{C}$ which is a contradiction.

□

Claim 3.7 *If $\mathcal{I}_{block}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $|\mathcal{P}(\text{PC} \in \{27.5, 28.1\})| = 1$ in $\mathcal{C}$. To violate $\mathcal{I}_{block}$, s would have to set V to *false* on Line 28.1. However, if s is a step from Line 28.1 to Line 28.2, then $|\mathcal{P}(\text{PC} \in \{27.5, 28.1\})| \neq 1$ in $\mathcal{C}.s$.
- Assume $|\mathcal{P}(\text{PC} \in \{27.5, 28.1\})| \neq 1 \wedge V =$ *false* in $\mathcal{C}$. To violate $\mathcal{I}_{block}$, s would have to step to Line 27.4 from Line 27.5 since $|\mathcal{P}(\text{PC} \in \{27.5, 28.1\})| = 0$ by $\mathcal{I}_{G2}$. However, s would then set V to *true*.

□

Claim 3.8 *If $\mathcal{I}_{wri}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $|\mathcal{P}(\text{PC} = 28.1)| = 1$ in $\mathcal{C}$. To violate $\mathcal{I}_{wri}$, s must set U to CLEAR or UPGRADING. U can be set to CLEAR on Line 16.1, but only if it's value is UPGRADING, so s can not be a step at Line 16.1. U can also be set to UPGRADING and CLEAR at Line 27.2 and Line 28.2, respectively. However, by $\mathcal{I}_{G2}$, there can be no process at either of these lines.
- Assume $|\mathcal{P}(\text{PC} = 28.1)| \neq 1 \wedge U \neq$ WRITING in $\mathcal{C}$. To violate $\mathcal{I}_{wri}$, s must be a step to Line 28.1. However, a process can only step to Line 28.1 from Line 27.5 and stepping from Line 27.5 to Line 28.1 will set U to WRITING.

□

Claim 3.9 *If $\mathcal{I}_{uu}$ holds in $\mathcal{C}$, then it also holds in $\mathcal{C}.s$.*

PROOF. By cases:

- Assume $U =$ UPGRADING $\wedge |\mathcal{P}(PC \in \{27.4, 27.5\})| = 1$ in $\mathcal{C}$. To violate $\mathcal{I}_{uu}$, s must be a step from Line 16.1 to Line 17. Such a step will set U to CLEAR.

- Assume $U \neq$ UPGRADING $\wedge\, |\mathcal{P}(PC \in \{27.4, 27.5\})| = 1 \wedge \mathcal{R}(\text{PC} \in \{17 \ldots 22.2\}) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{uu}$, s must set U to UPGRADING. U can only be set to UPGRADING on Line 27.2, and by $\mathcal{I}_{G2}$, there can be no process on Line 27.2.

- Assume $U =$ UPGRADING $\wedge\, |\mathcal{P}(PC \in \{27.4, 27.5\})| \neq 1 \wedge \mathcal{R}(\text{PC} \in \{17 \ldots 22.2\}) \neq \emptyset$ in $\mathcal{C}$. By $\mathcal{I}_{G2}$, $|\mathcal{P}(PC \in \{27.4, 27.5\})| = 0$. To violate $\mathcal{I}_{uu}$, s must be a step from Line 27.3 to Line 27.4. For this to be possible C must be less than or equal to 1. Therefore, by $\mathcal{I}_G$, $\mathcal{R}(\text{PC} \in \{17 \ldots 22.2\}) = \emptyset$.

□

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

PROOF. By cases:

- Assume $\mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) \neq \emptyset \wedge U =$ CLEAR in $\mathcal{C}$. To violate $\mathcal{I}_{read}$, s must set U to a value other than CLEAR. The only place where this can happen is Line 27.2 (the CAS at Line 27.5 will fail since $U =$ CLEAR). But if a process is at Line 27.2, there can be no process at Line 27.4 or 27.5 by $\mathcal{I}_{G2}$.

- Assume $\mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) \neq \emptyset \wedge \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) = \emptyset$ in $\mathcal{C}$. To violate, $\mathcal{I}_{read}$, s must be a step from Line 27.3 to Line 27.4. However, by $\mathcal{I}_G$, $C > 1$ so such a step is impossible.

- Assume $\mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) = \emptyset \wedge U \neq$ CLEAR $\wedge\, \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{read}$, s must be a step by a reader from Line 22.1, 22.2, or 28.2 to Line 23 or 27.1 or a step by the writer from Line 25.3 or 28.2 to Line 26.1 or Line 27.1.

  We will first consider a step by a reader or the writer from Line 28.2. In this case, U will be set to CLEAR, so the invariant will hold.

  We will next consider the case of a step by a reader from Line 22.1 or 22.2. By the contrapositive of $\mathcal{I}_{uu}$, $U \neq$ UPGRADING $\vee\, |\mathcal{P}(PC \in \{27.4, 27.5\})| \neq 1$. In the latter case, $\mathcal{P}(PC \in \{27.4, 27.5\}) = \emptyset$ by $\mathcal{I}_{G2}$. In the former case, either $U =$ CLEAR (precluded by our assumption) or $U =$ WRITING. If $U =$ WRITING, then by $\mathcal{I}_{uw}$ and $\mathcal{I}_{G2}$, there can be no process at Line 27.4 or 27.5 which is a contradiction.

  Finally, we consider the case of a step by the writer from 25.3. By item 6 of $\mathcal{I}_{25.3}$, $\mathcal{R}(PC \in \{27.4, 27.5\}) = \emptyset$ in $\mathcal{C}$. So even if the writer takes a step, the invariant holds.

□

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

PROOF. By cases:

- Assume $U = \text{WRITING} \wedge V = \textit{true}$ in $\mathcal{C}$. To violate $\mathcal{I}_{uwvt}$, s must be a step by a reader from Line 22.1, 22.2, or 28.2 to Line 23 or Line 27.1 or a step by the writer from Line 25.3 or 28.2 to Line 26.1 or 27.1.

  We first consider the case of a step by a reader or the writer from Line 28.2. In this case, U will be set to CLEAR, so the invariant will hold.

  We next consider a step by a reader from 22.1 or 22.2. The reader obviously cannot step to 23 or 27.1, so the invariant holds.

  Finally, we consider the case of a step by the writer from 25.3. By item 6 of $\mathcal{I}_{25.3}$, $\mathcal{R}(\text{PC} \in \{28.1 \ldots 28.2\}) = \emptyset$ and therefore, by the contrapositive of $\mathcal{I}_{uw}$, $U \neq \text{WRITING}$ which is a contradiction.

- Assume $U = \text{WRITING} \wedge V = \textit{false} \wedge \mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{uwvt}$, s must set V to *true*. This can only happen on Line 27.4. By $\mathcal{I}_{read}$, $\mathcal{P}(\text{PC} \in \{27.4, 27.5\}) = \emptyset$, so this is impossible.

- Assume $U \neq \text{WRITING} \wedge V = \textit{true} \wedge \mathcal{R}(\text{PC} = 23) \cup \mathcal{W}(\text{PC} = 26.1) \cup \mathcal{P}(\text{PC} = 27.1) \neq \emptyset$ in $\mathcal{C}$. To violate $\mathcal{I}_{uwvt}$, s must set U to WRITING. This can only happen on line 27.5. By $\mathcal{I}_{read}$, $U = \text{CLEAR} \vee \mathcal{P}(\text{PC} \in \{27.4, 27.5\}) = \emptyset$, so either there is no process on Line 27.5 or the CAS on Line 27.5 will fail.

□

Before we prove the invariants case by case based on the value of PC_w and the values of some shared variables, we make the following simple observation based on the code of `Reader-Try`. It simply says that a step by a reader cannot change the values of $D, Gate$ and PC_w.

Proposition 4.1 *If s is a step by a reader, then s does not change D,$Gate$ or PC_w.*

We also make the following observations about the variable X and the set of processes which can change it. One can easily verify these observations by simple inspection of Lines 9 through 14, Lines 18 through 20 and Line 8.

Proposition 4.2 *X is only set to true at Line 14, and if $X =$* true*, then it does not change till the writer executes Line 8.*

From now on we will assume these propositions as mere facts of the algorithm.

Claim 3.12 *If $\text{PC}_w \in \{1, 2, 3\}$ in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$*

PROOF. As $\text{PC}_w \in \{1, 2, 3\}$ in $\mathcal{C}$, so we know that $\mathcal{I}_{1,2,3}$ holds in $\mathcal{C}$.

We will prove this claim and all the subsequent claims in this lemma based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: As s cannot change PC_w we have to essentially show that $\mathcal{I}_{1,2,3}$ is satisfied in $\mathcal{C}.s$. As s cannot change $Gate$, Item 1 of $\mathcal{I}_{1,2,3}$ will remain unaffected. So we only need to show that Items 2-4 of $\mathcal{I}_{1,2,3}$ are still satisfied in $\mathcal{C}.s$. Item 4 of $\mathcal{I}_{1,2,3}$ still holds because by Item 3 of $\mathcal{I}_{1,2,3}$, no reader is at Line 15 in $\mathcal{C}$. As $\mathcal{R}(PC = 14) = \emptyset$ in $\mathcal{C}$,

$X \neq$ *true* in $\mathcal{C}.s$. And as $Permit =$ *true* in $\mathcal{C}$, no reader can execute Line 12 and enter the set $\mathcal{R}(PC \in \{13, 14\})$ in $\mathcal{C}.s$. Combining the previous two facts one can see that Item 2 of $\mathcal{I}_{1,2,3}$ still holds in $\mathcal{C}.s$. Item 3 will still hold because there is no reader at Line 14 in $\mathcal{C}$ (Item 2 of $\mathcal{I}_{1,2,3}$) and no reader can execute Line 17 and proceed to Line 18 with $d = \overline{D}$.

- s is the step by the writer. As $\text{PC}_w \in \{1, 2, 3\}$, so the writer can either execute Line 1 and move to Line 2, or execute Line 2 and move to Line 3, or execute Line 3 and move to Line 4. The first two cases are not interesting as the writer does not change any shared variables and PC_w is still in $\{1, 2, 3\}$, hence all Items of $\mathcal{I}_{1,2,3}$ still hold.

  Now say the writer executes Line 3, then $\text{PC}_w = 4$ in $\mathcal{C}.s$. Hence, we have to show that $\mathcal{I}_4$ is satisfied in $\mathcal{C}.s$. One can easily see that Items 2-4 of $\mathcal{I}_4$ are implied by the fact the Items 2-4 of $\mathcal{I}_{1,2,3}$ hold in $\mathcal{C}$. Item 1 of $\mathcal{I}_4$ holds in $\mathcal{C}.s$ because at Line 4, the writer toggles D and as $Gate = D$ in $\mathcal{C}$, $Gate = \overline{D}$ in $\mathcal{C}.s$.

□

Claim 3.13 *If* $\text{PC}_w = 4$ *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF. As $\text{PC}_w = 4$ in $\mathcal{C}$, we know that $\mathcal{I}_4$ holds in $\mathcal{C}$.
We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: We have to show that $\mathcal{I}_4$ still holds in $\mathcal{C}.s$. One can easily verify that items of $\mathcal{I}_4$ still hold in $\mathcal{C}.s$ because of the same reason they held in $\mathcal{I}_{1,2,3}$.

- s is a step by the writer: If the writer takes the step s, it will proceed to Line 5 (or simply Line 9). Also note that as $X \neq$ *true* in $\mathcal{C}$ and the writer does not change X in Line 4, so $X \neq$ *true* in $\mathcal{C}$. So we have to verify that $\mathcal{I}_9$ holds in $\mathcal{C}.s$. The only shared variable the writer changes at Line 4 is $Permit$ (it sets it *false*). So Item 3 of $\mathcal{I}_9$ trivially holds in $\mathcal{C}.s$. As there are no processes at Line 14 in $\mathcal{C}$, one can see that $\mathcal{PID}(14) = \emptyset$ in $\mathcal{C}.s$. Hence, Item 4 of $\mathcal{I}_9$ holds in $\mathcal{C}.s$. One can easily that rest of the items in $\mathcal{I}_9$ are simply implied by the fact that $\mathcal{I}_4$ holds in $\mathcal{C}$.

□

Claim 3.14 *If* $\text{PC}_w = 5$ *(or,* $\text{PC}_w = 9$*) and* $X \neq$ true *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF. As $\text{PC}_w = 9 \wedge X \neq$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_9$ holds in $\mathcal{C}$.
We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: We know s cannot change PC_w but it can change X from some pid to *true*. So we will prove this part by further two cases depending upon whether step s changes X or not.

1. s does set X to *true*: As s cannot change Item 1 of $\mathcal{I}_9$, we only need to show that Items 2-4 of $\mathcal{I}_9$ are still satisfied in $\mathcal{C}.s$. As Item 2 of $\mathcal{I}_9$ ($\mathcal{R}(PC = 15) = \emptyset$) holds in $\mathcal{C}$, to violate it in $\mathcal{C}.s$, s has to be a step at Line 14 such that r proceeds to Line 15 after it. But this would mean that X is changed to *true*, so $\mathcal{I}_9$ would no longer apply. As there is no reader at Line 15 in $\mathcal{C}$, Item 3 of $\mathcal{I}_9$ still holds in $\mathcal{C}.s$.

   Now we come to the most interesting case; say Item 4 of $\mathcal{I}_9$ is violated in $\mathcal{C}.s$. This can happen by two cases ; (a) if s is a step from Line 13 to Line 14 and the condition $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$ is not true in $\mathcal{C}.s$, (b) $X \in \mathcal{PID}(14)$ in $\mathcal{C}$ and $\mathcal{C}.s$, and condition $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$ is violated by s. If the case (a), where s is a step from Line 13 to Line 14, C should be 0 in $\mathcal{C}$. Hence, by $\mathcal{I}_G$, $\mathcal{R}(PC \in \{16.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset$ in $\mathcal{C}$ and $\mathcal{C}.s$. This would trivially imply $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$.

   So now lets come to the more interesting case, i.e., case (b), when $X \in \mathcal{PID}(14)$ in $\mathcal{C}$ as well as $\mathcal{C}.s$, but the step s violates the condition $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{21, 22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$. We will consider each part of this condition separately in the following cases :

   - $X \in \mathcal{PID}(14) \wedge \mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \neq \emptyset$ in $\mathcal{C}.s$: As Line 17 is $d \leftarrow D$, and $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) = \emptyset$ in $\mathcal{C}$, one can clearly see that no reader can enter the set $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\})$ in step s.
   - $X \in \mathcal{PID}(14) \wedge \mathcal{R}(PC \in \{21 \dots 23, 27.1 \dots 29.2\}) \neq \emptyset$ in $\mathcal{C}.s$: This case is further broken down in two cases, based on whether r enters Line 22.1 or Line 21 in s. Say r enters Line 22.1 in step s. There is no process at Line 21 in $\mathcal{C}$, so it means that s is a step at Line 22. As $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) = \emptyset$ in $\mathcal{C}$, one can see that if r takes a step at Line 22 (**wait till** $Gate = d$), it will have $d = D$. But in this case r cannot be at Line 22.1, because $Gate = \overline{D}$ in $\mathcal{C}$.

     Now say r enters Line 21 by step s. It cannot enter Line 21 from Line 19 because $X \in \mathcal{PID}$ in $\mathcal{C}$. As conditions for Item 4 hold in $\mathcal{C}$, so $\mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$. Hence, $r.x = X$ in $\mathcal{C}$. So if r enters Line 21 from Line 20, X will be equal to $r.pid$ in $\mathcal{C}.s$. Hence, $X \in \mathcal{PID}(21)$ in $\mathcal{C}.s$, which is a contradiction.
   - $X \in \mathcal{PID}(14) \wedge \mathcal{R}(\text{PC} \in \{19, 20\}) \notin \mathcal{R}(x = X)$: As $\mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$ in $\mathcal{C}$, so $\mathcal{R}(\text{PC} \in \{19, 20\}) \in \mathcal{R}(x = X)$ in $\mathcal{C}.s$ also, because if r takes a step at Line 18 ($x \leftarrow X$), it will enter Line 19 with $x = X$.

2. s sets X to *true*: As $Permit =$ *false* in $\mathcal{C}$ and s changes X to *true*, we have to show that $\mathcal{I}_{ptf}$ holds in $\mathcal{C}.s$. One can easily see that Item 1 of $\mathcal{I}_{ptf}$ holds. As s sets X to *true*, it means that s is a step at Line 14 (CAS$(X, i,$ *true*$)$). Moreover, s is a successful CAS. Hence, r is at Line 15 in $\mathcal{C}.s$, and as there were no readers at Line 15 in $\mathcal{C}$ (Item 2 of $\mathcal{I}_9$), r is the only process at Line 15 in $\mathcal{C}.s$, hence Item 2 of $\mathcal{I}_{ptf}$ holds in $\mathcal{C}.s$.

   As r succeeds in the CAS at Line 14 (CAS$(X, r.pid, tr)$), it means that $X \in \mathcal{PID}(14)$ in $\mathcal{C}$. By Item 4 of $\mathcal{I}_9$, this means that $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{22.1 \dots 23, 27.1 \dots 29.2\}) = \emptyset$ in $\mathcal{C}$. This means that $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) \cup \mathcal{R}(\text{PC} \in \{22.1 \dots 23, 27.1 \dots 29.2\})$ is also empty in $\mathcal{C}.s$, hence Item 3 of $\mathcal{I}_{ptf}$ also holds in $\mathcal{C}.s$.

- s is the step by the writer. If the writer takes the step s, it proceeds to Line 10. So we have to show that $\mathcal{I}_{10,11}$ holds in $\mathcal{C}.s$. Item 1-4 of $\mathcal{I}_{10,11}$ are exactly identical to Item 1-4 of $\mathcal{I}_9$. And as the writer does not change any of those items in s, they will continue to hold in $\mathcal{C}.s$. Item 5 of $\mathcal{I}_{10,11}$ is trivially true in $\mathcal{C}.s$, because when the writer executes Line 9, it sets $w.x$ to X.

□

Claim 3.15 *If* $PC_w \in \{10, 11\} \wedge X \neq$ true *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF. As $\text{PC}_w \in \{10, 11\} \wedge X \neq$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_{10,11}$ holds in $\mathcal{C}$.

We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: Again we have two cases here depending upon whether s changes X to *true* or not.

  1. s changes X to *true*: If s changes X to *true*, we have to show that $\mathcal{I}_{ptf}$ holds in $\mathcal{C}.s$. The proof of this part is exactly same as in the case when $\text{PC}_w = 9 \wedge X \neq$ *true* in $\mathcal{C}$ and a reader takes a step s to change X to *true*.
  2. s does not change X to *true*: So we have to verify that $\mathcal{I}_{10,11}$ still holds in $\mathcal{C}.s$. The proof that Items 1-4 of $\mathcal{I}_{10,11}$ still hold in $\mathcal{C}.s$, is exactly same as the case when $\text{PC}_w = 9 \wedge X \neq$ *true* in $\mathcal{C}$ and a reader takes a step s that does not set X to *true*.

     Now we have to show that Item 5 of $\mathcal{I}_{10,11}$ still holds $\mathcal{C}.s$. To see why this is true, note that there is a continuity in the sets $\mathcal{R}(\text{PC} \in \{16.1 \ldots 24, 9, 27.1 \ldots 29.2\})$, $\mathcal{R}(x = X, \text{PC} \in \{10, 11\})$ and $\mathcal{PID}(12 \ldots 14)$. By continuity we mean that if r leaves the set $\mathcal{R}(\text{PC} \in \{16.1 \ldots 24, 9, 27.1 \ldots 29.2\})$, then r enters the set $\mathcal{R}(x = X, \text{PC} \in \{10, 11\})$. Similarly, when a r leaves the set $\mathcal{R}(x = X, \text{PC} \in \{10, 11\})$, r enters Line 12 with $X = r.x$, hence $X \in \mathcal{PID}(12 \ldots 14)$. Hence, the only way s can violate Item 5 of $\mathcal{I}_{10,11}$ if r executes Line 14 (CAS$(X, r.pid,$ *true*$)$) while $X = r.pid$ in $\mathcal{C}$. But then s will set X to *true*, which is a contradiction.

- s is a step by the writer: As Line 10 is just a local step, then all the invariants trivially hold. So assume s is a step at Line 11 (CAS(X, x, i)). There are two cases depending upon whether the CAS at Line 11, succeeds or not.

  1. CAS in step s succeeds: Then $\text{PC}_w = 12$ and $X = w.pid$ in $\mathcal{C}.s$, hence we have to verify that $\mathcal{I}_{12\ldots14}$ holds in $\mathcal{C}.s$. Items 1-3 of $\mathcal{I}_{12\ldots14}$ hold in $\mathcal{C}.s$ because Items 1-3 of $\mathcal{I}_{10,11}$ hold in $\mathcal{C}$. Items 4,5 of $\mathcal{I}_{12\ldots14}$ trivially hold because $X = w.pid \wedge \text{PC}_w = 12$ in $\mathcal{C}.s$.
  2. CAS in step s does not succeed: This means that $X \neq w.x$ in $\mathcal{C}$ and $\text{PC}_w = 6$ in $\mathcal{C}.s$. As $X \neq$ *true* in $\mathcal{C}$ and s does not change X, so $X \neq$ *true* in $\mathcal{C}.s$. So we have to verify that $\mathcal{I}_6$ holds in $\mathcal{C}.s$. Item 4 of $\mathcal{I}_6$ holds in $\mathcal{C}.s$ because Item 4 of $\mathcal{I}_{10,11}$ holds in $\mathcal{C}$. Item 5 of $\mathcal{I}_6$ holds in $\mathcal{C}.s$ because $X \neq w.x$ in $\mathcal{C}$ and Item 5 of $\mathcal{I}_{10,11}$ holds in $\mathcal{C}$. Items 1-3 of $\mathcal{I}_6$ are simply implied by the fact that Items 1-3 of $\mathcal{I}_{10,11}$ hold in $\mathcal{C}$.

□

Claim 3.16 *If PC$_w \in \{12 \ldots 14\} \wedge X \neq$ true in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$*

PROOF. As PC$_w \in \{12 \ldots 14\} \wedge X \neq$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_{12\ldots14}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: The proof this is identical to the case when PC$_w \in \{10, 11\} \wedge X \neq$ *true*.

- s is the step by the writer: If s is at Line 12 (if $\neg Permit$), by Item 3 of $\mathcal{I}_{12\ldots14}$, $Permit = f$ in $\mathcal{C}$, hence the writer will proceed to Line 13. If s is at Line 13, then depending upon the value of C, the writer moves to Line 14 or Line 6. If it moves to Line 14, as no shared variables have changed from $\mathcal{C}$, $\mathcal{I}_{12\ldots14}$ will still hold in $\mathcal{C}.s$. If $C > 0$ in $\mathcal{C}$, then PC$_w = 6$ in $\mathcal{C}.s$. As X is still not *true* in $\mathcal{C}.s$, then we have to verify that $\mathcal{I}_6$ holds in $\mathcal{C}.s$. Items 1-4 of $\mathcal{I}_6$ still hold in $\mathcal{C}.s$ because they held in $\mathcal{C}$ and s did not change any shared variables at Line 13. As $C > 0$ in $\mathcal{C}$, by $\mathcal{I}_G$, $\mathcal{R}(PC \in \{16.1 \ldots 23, 27.1 \ldots 29.2\}) \neq \emptyset$ in $\mathcal{C}.s$. Hence, Item 5 of $\mathcal{I}$ also holds in $\mathcal{C}.s$.

  Now say s is at Line 14 (CAS$(X, i,$ *true*$)$), then there are two cases depending upon CAS in s succeeds or not. We examine both of these cases below.

  1. CAS in step s succeeds: Then PC$_w = 15$ and $X =$ *true* in $\mathcal{C}.s$, hence we have to verify that $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$. Items 1-3 of $\mathcal{I}_{15}$ hold in $\mathcal{C}.s$ because Items 1-3 of $\mathcal{I}_{12\ldots14}$ hold in $\mathcal{C}$. Item 5 of $\mathcal{I}_{12\ldots14}$ trivially holds in $\mathcal{C}.s$ because $X =$ *true* in $\mathcal{C}.s$. As the CAS at Line 14 succeeds in step s, it means $X = w.pid$ in $\mathcal{C}$. Hence by the fact that Item 5 of $\mathcal{I}_{12\ldots14}$ holds in $\mathcal{C}$, one can see that Item 4 of $\mathcal{I}_{15}$ will also hold in $\mathcal{C}.s$.
  2. CAS in step s does not succeed: This means that $X \neq w.pid$ in $\mathcal{C}$ and PC$_w = 6$ in $\mathcal{C}.s$. As $X \neq$ *true* in $\mathcal{C}$ and s does not change X, so $X \neq$ *true* in $\mathcal{C}.s$. So we have to verify that $\mathcal{I}_6$ holds in $\mathcal{C}.s$. Item 5 of $\mathcal{I}_6$ holds in $\mathcal{C}.s$ because $X \neq w.pid$ in $\mathcal{C}$ and Item 5 of $\mathcal{I}_{12\ldots14}$ holds in $\mathcal{C}$. Items 1-4 of $\mathcal{I}_6$ still hold in $\mathcal{C}.s$ because Items 1-4 of $\mathcal{I}_{12\ldots14}$ hold in $\mathcal{C}$ and s did not change any shared variables.

□

Claim 3.17 *If PC$_w \in \{9 \ldots 14\} \wedge X =$ true $\wedge\ Permit =$ false in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$*

PROOF.

As PC$_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\ Permit =$ *false* in $\mathcal{C}$, we know that $\mathcal{I}_{ptf}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: We have further two cases depending upon whether s changes $Permit$ to *true* or not. We will consider these two cases separately.

  1. s does not set $Permit$ to *true*: So we have to verify that $\mathcal{I}_{ptf}$ still holds in $\mathcal{C}.s$. Item 1 of $\mathcal{I}_{ptf}$ still holds because a step by r cannot change $Gate$. As s does not set $Permit$ to *true*, it means that r did not execute Line 15 in s. Also, there is only one reader at

Line 15 in $\mathcal{C}$, combining the previous two facts together we get $|\mathcal{R}(\text{PC} = 15)| = 1$ in $\mathcal{C}.s$. Hence, Item 2 of $\mathcal{I}_{ptf}$ still holds in $\mathcal{C}.s$.

Now say Item 3 $\mathcal{I}_{ptf}$ is violated in $\mathcal{C}.s$. This means that in step s either r executed Line 17 and entered Line 18 with $d = \overline{D}$, or r has side D and it enters Line 22.1. The former is not possible because Line 17 is $d \leftarrow D$.

Now say r enters Line 22.1 in step s. This means s is a step at Line 21 or at Line 22. If s is a step at Line 21, then r cannot be at Line 22.1 in $\mathcal{C}.s$ as $X =$ *true* in $\mathcal{C}$. As $\mathcal{R}(d = \overline{D}, \text{PC} \in \{18 \dots 22\}) = \emptyset$ holds in $\mathcal{C}$, one can see that if r takes a step at Line 22 (**wait till** $Gate = d$), it will have $d = D$. But in this case r cannot be at Line 22.1, because $Gate = \overline{D}$ in $\mathcal{C}$.

2. s sets $Permit$ to *true*: This means that r is at Line 15 in $\mathcal{C}.s$, and as Item 2 of $\mathcal{I}_{ptf}$ holds in $\mathcal{C}$, it means that $\mathcal{R}(PC = 15) = \{r\}$ in $\mathcal{C}$. As $Permit$ is set to *true* in s, it means that r executes step s at Line 15. Hence, there is no other reader at Line 15 in $\mathcal{C}.s$, so Item 2 of $\mathcal{I}_{ptf}$ holds in $\mathcal{C}.s$. Item 1 and 2 of $\mathcal{I}_{ptf}$ are true in $\mathcal{C}.s$ for the same reason they are true in the previous case when the step s by r does not set $Permit$ to *true*.

- s is a step by the writer: Note that when the writer takes a step at Lines 9 through 14 and $X = tr$, it cannot change any shared variables. Hence, either it stays at Lines 9 through 14 and $\mathcal{I}_{ptf}$ trivially holds in $\mathcal{C}.s$. Or, it moves to Line 6 and $\mathcal{I}_{6tf}$ (which is identical to $\mathcal{I}_{ptf}$) holds in $\mathcal{C}.s$.

□

Claim 3.18 *If* $PC_w \in \{9 \dots 14\} \wedge X =$ true $\wedge\, Permit =$ true *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w \in \{9 \dots 14\} \wedge X =$ *true* $\wedge\, Permit =$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_{ptt}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: s cannot change PC_w and by item 2 of $\mathcal{I}_{ptt}$ we know s cannot set $Permit$ to *false*. So we have to verify that $\mathcal{I}_{ptt}$ holds in $\mathcal{C}.s$.

  The proof of why Item 1 and 3 of $\mathcal{I}_{ptt}$ hold in $\mathcal{C}.s$ is exactly same as the case when s is a step by the reader which does not change $Permit$ when $\text{PC}_w \in \{9 \dots 14\} \wedge X =$ *true* $\wedge\, Permit =$ *false*. As $X =$ *true* in $\mathcal{C}$, so in step s, r cannot enter Line 15 by executing Line 14. Hence Item 2 of $\mathcal{I}_{ptt}$ holds in $\mathcal{C}.s$

- s is a step by the writer: Note that when the writer takes a step at Lines 9 through 14 and $X = tr$, it cannot change any shared variables. Hence, either it stays at Lines 9 through 14 and $\mathcal{I}_{ptt}$ trivially holds in $\mathcal{C}.s$. Or, it moves to Line 6 and $\mathcal{I}_{6tt}$ (which is identical to $\mathcal{I}_{ptt}$) holds in $\mathcal{C}.s$.

□

Claim 3.19 *If* $PC_w = 15$ *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w = 15$ in $\mathcal{C}$, we know that $\mathcal{I}_{15}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: s cannot change PC_w, so we have to verify that $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$. Item 1 of $\mathcal{I}_{15}$ trivially holds in $\mathcal{C}.s$. As $X =$ *true* in $\mathcal{C}$, so in step s, r cannot enter Line 15 by executing Line 14. Hence Item 2 of $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$. Similarly, as r is not at Line 15 in $\mathcal{C}$, Item 3 of $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$.

  The proof of why Item 4 of $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$ is exactly same as the proof for Item 3 of $\mathcal{I}_{ptf}$, when s is a step by the reader which does not change $Permit$ when $\text{PC}_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\, Permit =$ *false*. As $X =$ *true* in $\mathcal{C}$, so in step s, r cannot change X, hence Item 5 of $\mathcal{I}_{15}$ holds in $\mathcal{C}.s$.

- s is a step by the writer: When the writer performs a step at Line 15, it will set $Permit$ to *true* and move to Line 6. As $X =$ *true* in $\mathcal{C}$, one has to show that $\mathcal{I}_{6tt}$ holds in $\mathcal{C}.s$. One can clearly see that Items 1, 2 and 4 of $\mathcal{I}_{15}$ imply Items 1,2 and 3 of $\mathcal{I}_{6tt}$ (of $\mathcal{I}_{ptt}$). And as s does not change any shared variables except $Permit$ in step s, $\mathcal{I}_{6tt}$ clearly holds in $\mathcal{C}.s$.

□

Claim 3.20 *If* $\text{PC}_w = 6 \wedge X \neq$ true *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w = 6X \neq$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_6$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: There are again two cases here based on whether X is set to *true* in s or not. If X is not set to *true* by s then we have to show that $\mathcal{I}_6$ still holds in $\mathcal{C}.s$. On the other hand if X is set to *true* by s then we have to show that $\mathcal{I}_{6tf}$ holds in $\mathcal{C}.s$. The arguments to prove both these cases are identical to the case when s is step by a reader and $\text{PC}_w \in \{10, 11\} \wedge X \neq$ *true*.

- s is a step by the writer: If the writer executes Line 6 (**wait till** $Permit$), it will not change the PC_w or any shared variable. Hence, $\mathcal{I}_6$ will trivially hold in $\mathcal{C}.s$.

□

Claim 3.21 *If* $\text{PC}_w = 6 \wedge X =$ true $\wedge\, Permit =$ false *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w = 6X \neq$ *true* $\wedge\, Permit =$ *false* in $\mathcal{C}$, we know that $\mathcal{I}_{6tf}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: There are again two cases here based on whether $Permit$ is set to *true* in s or not. If $Permit$ is not set to *true* by s then we have to show that $\mathcal{I}_{6tf}$ still holds in $\mathcal{C}.s$.

  On the other hand if $Permit$ is set to *true* by s then we have to show that $\mathcal{I}_{6tt}$ holds in $\mathcal{C}.s$. The arguments to prove both these cases are identical to the case when s is step by a reader and $\text{PC}_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\ Permit \neq$ *true*.

- s is a step by the writer: If the writer executes Line 6 (**wait till** $Permit$), it will not change the PC_w or any shared variable. Hence, $\mathcal{I}_6$ will trivially hold in $\mathcal{C}.s$.

□

Claim 3.22 *If* $\text{PC}_w = 6 \wedge X =$ true $\wedge\ Permit =$ true *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w = 6X \neq$ *true* $\wedge\ Permit =$ *true* in $\mathcal{C}$, we know that $\mathcal{I}_{6tt}$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: As there are no readers at Line 15 and $X =$ *true* in $\mathcal{C}$, the step s cannot change $Permit$ or X, hence we have to argue that $\mathcal{I}_{6tt}$ still holds in $\mathcal{C}.s$. The proof of this is identical to the case when s is step by a reader and $\text{PC}_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\ Permit =$ *true*.

- s is a step by the writer: In this case the writer will proceed to Line 7 or Line 25.1. So we have to show that $\mathcal{I}_7$ or $\mathcal{I}_{25.1/2}$ holds in $\mathcal{C}.s$. This is clearly true from the fact that $\mathcal{I}_{6tt}$ (or, $\mathcal{I}_{ptt}$) holds in $\mathcal{C}$ and the writer does not change any shared variables in s.

□

Claim 3.23 *If* $\text{PC}_w = 7$ *in* $\mathcal{C}$*, then* $\mathcal{I}$ *holds in* $\mathcal{C}.s$

PROOF.

As $\text{PC}_w = 7$ in $\mathcal{C}$, we know that $\mathcal{I}_7$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: We have to argue that $\mathcal{I}_7$ still holds in $\mathcal{C}.s$. The proof of this is identical to the case when s is a step by a reader and $\text{PC}_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\ Permit =$ *true* ($\mathcal{I}_{ptt}$).

- s is a step by the writer: In this case the writer will proceed to Line 8. So we have to show that $\mathcal{I}_8$ holds in $\mathcal{C}.s$. The writer sets the $Gate$ to D at Line 7. Hence, Item 1 of $\mathcal{I}_8$ holds in $\mathcal{C}.s$. Items 2-5 of $\mathcal{I}_8$ are implied by the fact that Items 2-5 of $\mathcal{I}_7$ hold in $\mathcal{C}$.

□

Claim 3.24 *If PC$_w = 8$ in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$*

PROOF.

As PC$_w = 8$ in $\mathcal{C}$, we know that $\mathcal{I}_8$ holds in $\mathcal{C}$. We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: We have to argue that $\mathcal{I}_8$ still holds in $\mathcal{C}.s$. As no process is at Line 15 in $\mathcal{C}$, so $Permit$ will still remain *true* in $\mathcal{C}.s$.

  As $X =$ *true* in $\mathcal{C}$, r cannot do a successful CAS on X and proceed to Line 15. Hence, Item 2 and Item 4 of $\mathcal{I}_8$ still hold. Rest of the proof is identical to the case when s is step by a reader and PC$_w \in \{9 \ldots 14\} \wedge X =$ *true* $\wedge\, Permit =$ *true* ($\mathcal{I}_{ptt}$).

- s is a step by the writer: In this case the writer will proceed to Line 1. So we have to show that $\mathcal{I}_{1,2,3}$ holds holds in $\mathcal{C}.s$. The writer sets the X to $w.pid$ at Line 8. And as no process has $pid =$ *true*, Item 2 of $\mathcal{I}_{1,2,3}$ holds in $\mathcal{C}.s$. Items 1,3,4 of $\mathcal{I}_{1,2,3}$ hold in $\mathcal{C}.s$ because they hold in $\mathcal{C}$ and the writer only changes X in s.

□

Claim 3.25 *If PC$_w \in \{25.1, 25.2\}$ in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

PROOF. As PC$_w \in \{25.1, 25.2\}$ in $\mathcal{C}$, so we know that $\mathcal{I}_{25.1/2}$ holds in $\mathcal{C}$.

We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: The argument is similar to the argument for $\mathcal{I}_7$.

- s is a step by the writer: In this case the writer will proceed to Line 25.2 (trivial) or Line 25.3. In the case of s moving to Line 25.3, the argument is similar to the argument for $\mathcal{I}_7$. As for the additional item that $\mathcal{R}(\text{PC} \in \{27.2 \ldots 29.2\}) = \emptyset$, it is obvious from item 4 of $\mathcal{I}_{25.1/2}$.

□

Claim 3.26 *If PC$_w = 25.3$ in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

PROOF. As PC$_w = 25.3$ in $\mathcal{C}$, so we know that $\mathcal{I}_{25.3}$ holds in $\mathcal{C}$.

We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: The argument is similar to the argument for $\mathcal{I}_8$. As for the additional item that $\mathcal{R}(\text{PC} \in \{27.2 \ldots 29.2\}) = \emptyset$, it is true because any process entering that section of code would have to move from Line 27.1 to Line 27.2 which is impossible by $\mathcal{I}_G$.

- s is a step by the writer: In this case the writer will proceed to Line 26.1, so we must show $\mathcal{I}_{down}$ holds. Since $\mathcal{I}_{25.3}$ has the same properties as $\mathcal{I}_8$ and $\mathcal{I}_{down}$ has the same properties as $\mathcal{I}_{1,2,3}$, the argument is similar to the argument for $\mathcal{I}_8$.

□

Claim 3.27 *If PC$_w \in \{26.1 \ldots 29.2\}$ in $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

PROOF. As PC$_w \in \{26.1 \ldots 29.2\}$ in $\mathcal{C}$, so we know that $\mathcal{I}_{down}$ holds in $\mathcal{C}$.

We will prove this claim based on the whether s is a step by a reader or the writer.

- s is a step by a reader r: The argument is similar to the argument for $\mathcal{I}_{1,2,3}$.
- s is a step by the writer: In this case the writer will proceed either to another line within $\{26.1 \ldots 29.2\}$ or to Line 1. If the former, then the invariant is trivially true, since none of those lines changes any of the shared variables referenced in the invariant. If the latter, we have to show that $\mathcal{I}_{1,2,3}$ holds in $\mathcal{C}.s$. Since the items of $\mathcal{I}_{down}$ are identical to the items of $\mathcal{I}_{1,2,3}$ and since s does not change any of these items, $\mathcal{I}_{1,2,3}$ holds in $\mathcal{C}.s$.

□

Hence, covering all these cases we have shown that if $\mathcal{I}$ holds in $\mathcal{C}$ then it also holds in $\mathcal{C}.s$, for every possible step s. □

4.3 Proof of Theorem 1

As with the invariants, several of these lemmas are taken directly from Bhatt and Jayanti's proof with minor changes. The following lemmas are taken directly from Bhatt and Jayanti's proof with little or no change: Lemma 5, Lemma 6, Lemma 11.

The following lemmas are taken from from Bhatt and Jayanti's proof with significant modifications: Lemma 4, Lemma 10, Lemma 8, Lemma 12

Before examining the lemmas, note the doorway and waiting room for each process. For the writer, the doorway consists of lines 1 through 5, and the waiting room is line 6. For the readers, the doorway consists of lines 16 through 21, and the waiting room consists of lines 22 through 22.2.

Lemma 4 (Mutual Exclusion) *A reader r and a normal writer w cannot be in the CS together, and a distinct process p and an upgraded process p' cannot be in the CS together.*

PROOF. One can easily see from $\mathcal{I}_7$ that there is no reader r in the CS (PC$_r \in \{23, 27.1 \ldots 29.2\}$) when w is in the CS. What about p and p'? By the restrictions in Figure 3, PC$_{p'} = 28.1$. As a result, $\mathcal{I}_{wri}$ and $\mathcal{I}_{block}$ hold, which means $\mathcal{I}_{uwvt}$ holds. In conjunction with $\mathcal{I}_{G2}$, this means that p cannot be in the CS (PC$_p \in \{23, 26.1 \ldots 28.2\}$) when p' is in the CS. □

The following three lemmas will be useful to prove the rest of the properties.

Lemma 5 *If at time t, a reader r is at Line 22 and PC$_w \in \{1, 8\}$ then $Gate = D$ while r is at Line 22.*

PROOF. As $\text{PC}_r = 22 \wedge \text{PC}_w \in \{1, 8\}$, so one can see from Item 3 of $\mathcal{I}_{1,2,3}$ and Item 4 of $\mathcal{I}_8$ that variable d of r is equal to D. Also by Item 1 of $\mathcal{I}_{1,2,3}$ and $\mathcal{I}_8$, $Gate = D$. W.L.O.G., let $d = D = 1$, to prove this lemma we will show that $Gate$ is not changed while r is at Line 22.

Say $Gate$ is set to 0, while r is still at Line 22. It means w executes $Gate \leftarrow 0$ (Line 7) at some time after t, such that $PC_r = 22, D = 0$ at t. But by Item 4 of $\mathcal{I}_7$, $\mathcal{R}(d = 1, \text{PC} \in \{18 \dots 23\}) = \emptyset$, this is a contradiction. □

Lemma 6 *If at time t, a reader r is at Line 22 and some reader is in the CS, then r will advance to Line 22.1 the next step it takes after t.*

PROOF. We will prove this lemma by two different cases based on the value of X at t.

If at time t, $X =$ *true* and a reader is in the CS (Line 23), from the inspection of all the invariants one can clearly see that $\text{PC}_w = 8$ at t. So from the Lemma 5, $Gate = r.d$ and when r steps, it will advance to Line 22.1.

Now assume, $X \neq$ *true* at t. We prove this case by the following claim, thereby concluding the Lemma.

Claim 6.1 *If at time t a reader r is at Line 22 and $X \neq$* true*, then r will advance to Line 22.1 the next step it takes after t.*

PROOF. As $\text{PC}_r = 22$, it means r previously observed $X =$ *true* at Line 21. So if $X \neq$ *true* at time t, by the inspection of the code one can see that only the writer changes X when it is *true* and it does that at Line 8. Hence, it means w executed Line 8 after r executes Line 21 and before t. Hence, $\text{PC}_w = 8, \text{PC}_r = 22$ at some time before t. By the Lemma 5, $Gate = r.d$ and r will advance to Line 22.1 the next step it takes after t.

□

Lemma 7 *If a reader r is in the waiting room when another process p is executing the Upgrade section ($PC_p \in \{27.1 \dots 27.5\}$), either r enters the CS before p finishes its current execution of the Upgrade section or p fails in its current execution of the Upgrade section.*

PROOF. Assume r does not enter the CS before p finishes its current execution of the Upgrade section. Consider the following cases:

1. Consider the case where p is on Line 27.1, 27.2, or 27.3. By $\mathcal{I}_G$, $C > 1$ and will remain so until after p finishes Upgrade. Therefore, p will finish Upgrade with a result of failure.

2. Consider the case where p is on Line 27.4 or 27.5. At the most recent time t that p read C on Line 27.3, $C = 1$. $C = 1 \implies \text{PC}_r \notin \{16.1 \dots 22.2\}$ at time t (by $\mathcal{I}_G$). Since r is now in the waiting room, r must have executed Line 16.1 at some time t' after time t. Furthermore, no other process could have executed Line 27.2 after time t' but before p completed Upgrade (by $\mathcal{I}_{G2}$). Therefore, p cannot return from Upgrade with a result of success because the CAS on Line 27.5 will fail.

□

Lemma 8 (Unstoppable Reader Property) *If a reader r is in the waiting room ($PC_r \in \{22 \ldots 22.2\}$) at time t, then r is* CS*-enabled at t if any of the following holds*

1. *if a normal reader is in the* CS *or a downgraded process is in the* CS *at time t.*
2. *if an upgrading process is not guaranteed to succeed at its current execution of Upgrade at time t.*
3. *if no upgraded process or upgrading process is in the CS or Exit section, $r >_{rp} w$, and w is not in the CS or the Exit section.*

PROOF.

Claim 8.1 *If a normal reader, upgrading process, or downgraded process is in the CS, there is no process on Line 28.1.*

PROOF. Obvious from Lemma 4 (Mutual Exclusion). □

We will take each of the cases above in turn.

1. r could be on any of 22, 22.1, and 22.2:

   22: By Lemma 6, if r is on Line 22, r will advance to Line 22.1 in one step.

   22.1: Note that U can only be set to WRITING when a process completes the Upgrade section. Since a normal reader is in the CS, there can be no process at Lines 28.1 ... 28.2 by $\mathcal{I}_{G2}$. By the contrapositive of $\mathcal{I}_{uw}$, at time t, $U \neq$ WRITING and by Lemma 7, $U \neq$ WRITING until r enters the CS. Therefore, if r is on Line 22.1, r will enter the critical section in one step.

   22.2: Finally, consider the case where r is on Line 22.2. By the contrapositive of $\mathcal{I}_{vt}$, if there is no process on Line 28.1 (which we have by Claim 8.1), either there is no reader on Line 22.2 or $V =$ *false*. Since r is on Line 22.2, V must be *false*. By $\mathcal{I}_{stuck}$, as long as r remains on 22.2, no reader can execute 27.4 and hence V cannot be set to *true*. Since $V =$ *false* as long as r remains on 22.2, r will enter the CS in one step.

2. Same as case 1.

3. As $r >_{rp} w$, it means that either r doorway precedes w, or some process is in the CS when r is in the waiting room ($PC_r \in \{22 \ldots 22.2\}$). r could be on any of 22, 22.1, and 22.2:

   22: If r doorway precedes w, $PC_w = 1$ (since r's doorway doesn't end until 22), hence by Lemma 5, r should advance to Line 22.1 in its next step after t. If there is some time when $PC_r = 22$, w is in the Try section, and some other reader is in the CS, by Lemma 6, r should advance to Line 22.1 in its next step after t.

   22.1: Same as case 1.

   22.2: Same as case 1.

□

Lemma 9 (Reader Priority Property) *If $r >_{rp} w$, then r enters the CS before w.*

PROOF. If $r >_{rp} w$, there are two possibilities: either r doorway preceded w or there was some time when r was in the waiting room when another reader was in the CS and w was in the Try section. We will consider these two cases separately:

1. r doorway preceded w. In other words, at some time t, $\text{PC}_r \in \{22 \dots 22.2\}$ when $\text{PC}_w = 1$. By $\mathcal{I}_{1,2,3}$, at time t, $X \notin \mathcal{PID}(13, 14)$ and $\mathcal{R}(\text{PC} = 15) = \emptyset$. As long as r does not enter the CS, $C > 0$ by $\mathcal{I}_G$. Note also that X can only be set to a given $i \in \mathcal{PID}$ by process i. Therefore, any process currently at Line 14 is guaranteed to fail the CAS at 14 and will not advance to Line 15. Additionally, no process will be able to get past Line 13 until r enters the CS. Since w must execute Line 4 at some time after t but before entering the CS, $Permit$ will be set to *false* and w will get stuck at Line 6. Since no one will be able to execute Line 15 before r enters the CS and Line 15 is the only place where $Permit$ can be set to *true*, w cannot enter the CS before r.

2. At some time t, r was in the waiting room, w was in the Try section, and some other reader r' was in the CS. If at time t, r' is a normal reader or an upgrading reader not guaranteed to succeed at its current execution of Upgrade, then by Lemma 8, we have r is enabled. Therefore, by Lemma 4, w does not enter the CS before r.

   Assume that at time t, r' is an upgraded reader or a reader whose current execution of the Upgrade section is guaranteed to succeed. By Lemma 6, r will advance to Line 22.1 the next step it takes.

   Before w enters the CS, r' must leave the CS (by Lemma 4). At the time t' that r' begins executing its Exit section, there are no readers at Line 27.4 or Line 27.5 (by $\mathcal{I}_{G2}$). Additionally, no reader will be able to reach those lines until r enters the CS since $C > 1$ (by $\mathcal{I}_G$). As soon as r' executes the first line of its Exit section, r will be enabled to enter the CS since V will be *false* and no process will be able to set it to *true* until r enters the CS. Therefore, by Lemma 4, w does not enter the CS before r.

□

Before we prove Livelock freedom we show that no reader starves.

Lemma 10 *If a reader r is in the Try section and no process crashes, then r eventually enters the CS.*

PROOF. Suppose r stays in the Try section forever. Then we first claim that the writer w also stays in the Try section forever. This is true because, if r keeps taking steps then it will eventually complete its doorway. Now if w ever enters the Remainder section after r has completed the doorway, by Lemma 9, w cannot enter the CS before r. Therefore, both r and w will be in the Try section forever after some time t, which means $\text{PC}_r \in \{22 \dots 22.2\}, \text{PC}_w = 6$ and $Permit =$ *false* forever after t.

If r is at Line 22 and r continues to take steps, $X =$ *true* by Claim 6.1. So $\text{PC}_w = 6, X =$ *true* and $Permit =$ *false* forever after t. But looking at the invariant for this case ($\mathcal{I}_{6tf}$), one can see that some reader is at Line 15 and it will eventually execute Line 15 and set $Permit$ to *true*. Which is a contradiction to the fact that $Permit = f$ for all times after t. Therefore r cannot be stuck at 22 forever.

If r is stuck at 22.2, V must be *true*. By $\mathcal{I}_{vt}$, there must be a process p' at Line 28.1 who will set V to *false*. By $\mathcal{I}_{G2}$, there can be no other process p s.t. $\text{PC}_p \in \{27.2 \ldots 27.5\}$ when p' is at Line 28.1. Furthermore, any process to complete the Upgrade section before r enters the CS will fail on Line 27.1 because $C > 1$ by $\mathcal{I}_G$. Therefore, if r is stuck on 22.2, V will eventually be set to *false* and cannot be changed back to *true* until after r enters the CS. Therefore, r can't be stuck at 22.2 forever. □

Lemma 11 (Livelock freedom) *If some process is in the Try section and no process crashes, then some process enters the* CS *eventually.*

PROOF. In the previous lemma we have shown that no reader starves. So to prove this lemma, we have to show that if no reader is active for all times after some time (say t), then the writer cannot stay in the Try section forever.

Say the writer stays in the Try section forever after all time $t' > t$. As the writer can stay only stay at Line 6 in the Try section forever, it means that there is some time $t^* > t' > t$, such that for all times after t^*, $\text{PC}_w = 6, Permit = f$ and no readers are active.

We first claim that $X =$ *true* at t^*. Say $X \neq$ *true* at t^*, as $\text{PC}_w = 6$ at t^*, by Item 5 of $\mathcal{I}_6$, one can see that some reader should be active at t^*, which is a contradiction.

So $X \neq$ *true* $\wedge\ \text{PC}_w = 6 \wedge Permit =$ *false* at t^*. By Item 5 of $\mathcal{I}_{6tf}$ one can again see that some reader should be active, which is a contradiction. Hence, it means that in the absence of the readers, the writer cannot stay in the Try section forever. □

Lemma 12 (Constant RMR complexity) *The algorithm given in Figure 2 has $O(1)$ RMR complexity in CC model.*

PROOF.

As all procedures except `Reader-Try` and `Writer-Try` have a constant number of steps and the doorways within these sections have a constant number of steps, all we have to show is that the RMR complexity when the processes are spinning is a constant.

In case of `Writer-Try` the writer waits for $Permit$ to be *true* (Line 6). The only place a reader can change Permit is Line 15, and it will be changed to *true*. Therefore, the next time the writer reads the variable, it will step past Line 6.

In the `Reader-Try`, say a reader r is on Line 22. W.L.O.G., say r is waiting for $Gate$ to be set to 1. By the arguments similar to the proof of Lemma 5, we know that once the $Gate$ is set to 1 while r is still at Line 22, it will not change to 0. Also by the inspection of the algorithm one can see $Gate$ is never overwritten with the same value, more precisely, the writer writes alternating values (1 and 0) into the $Gate$. Combining the two facts together one can see that while r is at Line 22, only a single write operation is performed on $Gate$.

Say a reader r is on Line 22.2. By $\mathcal{I}_{stuck}$, no process can be at Line 27.4 or 27.5, and since $C > 0$ (by $\mathcal{I}_G$), no process will be able to reach Line 27.4 or 27.5, and hence no process will be able to set V to *true* until r steps past Line 22.2. If at time t, $V =$ *false*, then r will move past 22.2 next time it takes a step. If not, eventually V will be set to *false*. When r reads $V =$ *false*, it will step past Line 22.2, so it will only read V as *false* once. □

Bounded Exit (P2), Bounded Upgrade (P9′), and Bounded Downgrade (P10′) are self-evident. First-In-First-Enabled (P4) and Concurrent Entering (P5′) are implied by Unstoppable Reader Priority (RP2′). Finally, Upgradeability (P8′) is obvious from $\mathcal{I}_G$ and observation of the algorithm. Therefore, the algorithm satisfies all properties specified in Theorem 1.

5 Extension to Multi-Writer Algorithm

Bhatt and Jayanti proved that the Single-Writer algorithm can easily be extended to a Multi-Writer version by wrapping the Single-Writer writer lock in a Mutex Lock satisfying starvation freedom, FCFS, and bounded exit [2]. We will not reproduce that work here, since we do not need to make any modifications. We will only comment that a writer does not release the aforementioned Mutex Lock until after it has left the CS of the single-writer algorithm, even if it downgrades. With this care, Bhatt and Jayanti's method of creating a Multi-Writer lock applies to the algorithm in Figure 2.

6 Model Checking

As an additional method of verification of the correctness of the algorithm, we used TLA+ and the TLC model checker [7]. TLA allows for formal specification of algorithms and their properties. We implemented the algorithm using the PlusCal language which allows a user to implement a multi-process algorithm that can be easily converted to TLA [7]. We then formulated the Mutual Exclusion invariant in TLA and ran the TLC model checker with four processes, three readers and a writer. TLC tries every possible interleaving of processes executing the algorithm and ensures that certain specified invariants hold throughout the execution; in our case, we ensure Mutual Exclusion (P1′). TLC also has deadlock checking built in, so we also ensure Livelock Freedom (P6). After running for over two days and processing a billion distinct states (1,081,613,412 was the final tally), TLC had found no violations of Mutual Exclusion or situations where deadlock occurred–at this point, the program was terminated due to lack of time. While not a proof of correctness by any means, this check does provide additional confidence that the algorithm is correct.

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 '10*, pages 468–477, 2010.

[4] Vibhor Bhatt. *Reader-Writer Lock: Rigorous Formulations and Constant RMR Algorithms*. PhD thesis, Dartmouth College, January 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] Leslie Lamport. *A PlusCal User's Manual: C-Syntax Version 1.5*, April 2011.