6-1-2012

Abortable Reader-Writer Locks are No More Complex Than Abortable Mutex Locks

Zhiyu Liu
Dartmouth College

Abortable Reader-Writer Locks are No More Complex Than Abortable Mutex Locks

Dartmouth Computer Science Technical Report TR2012-718

Zhiyu Liu

M.S. Thesis

Advisor: Prasad Jayanti

June 2012

Abstract

When a process attempts to acquire a mutex lock, it may be forced to wait if another process currently holds the lock. In certain applications, such as real-time operating systems and databases, indefinite waiting can cause a process to miss an important deadline [20]. Hence, there has been research on designing *abortable* mutual exclusion locks, and fairly efficient algorithms of $O(\log n)$ RMR complexity have been discovered [11, 14] (n denotes the number of processes for which the algorithm is designed).

The abort feature is just as important for a reader-writer lock as it is for a mutual exclusion lock, but to the best of our knowledge there are currently no abortable reader-writer locks that are starvation-free. We show the surprising result that any abortable, starvation-free mutual exclusion algorithm of RMR complexity $t(n)$ can be transformed into an abortable, starvation-free reader-writer exclusion algorithm of RMR complexity $O(t(n))$. Thus, we obtain the first abortable, starvation-free reader-writer exclusion algorithm of $O(\log n)$ RMR complexity. Our results apply to the Cache-Coherent (CC) model of multiprocessors.

Acknowledgments

First, I would like to thank my advisor, Professor Prasad Jayanti. It was you who brought me into the world of distributed algorithms and showed me how beautiful this world is. You have taught me a lot: enjoying the process of thinking, being brave in the uncertainty of research, thinking rigorously while keeping imagination, and so on. It has always been

amazing to work with you. I am so fortunate to have you as my advisor.

I would also like to thank my thesis committee members, Professor Thomas Cormen and Professor Robert Drysdale. Thank you for your invaluable comments and feedback on my thesis work.

Finally, I would like to thank my loving parents who always trust me and encourage me. thank you so much for everything you have done for me.

Contents

List of Figures

1 Introduction

1.1 Reader-Writer Exclusion

Mutual Exclusion, where n asynchronous processes share a resource that can be accessed by only one process at a time, is a fundamental problem in distributed computing [7]. In the standard formulation of this problem, each process repeatedly cycles through four sections of code—the *Remainder*, *Try*, *Critical*, and *Exit* Sections. The process stays in the *Remainder Section* as long as it is not interested in the resource. When it becomes interested, it executes the *Try Section* to compete with other process for the access to the resource. The process then enters the *Critical Section* (CS), where it accesses the resource. Finally, to relinquish its right over the resource, the process executes the *Exit Section* and moves back to the Remainder Section. The *mutual exclusion problem* is to design the Try and Exit Sections so that at most one process is in the CS at any time. The Try and Exit Sections are normally thought of as the acquisition and the release of an *exclusive lock* to the resource.

Reader-Writer Exclusion is an important and natural generalization of mutual exclusion. This problem was first formulated and solved over forty years ago by Courtois, Heymans, and Parnas [6] and continues to receive much research attention [18, 10, 3, 15, 5, 4]. Here the shared resource is a buffer and processes are divided into *readers* and *writers*. If a writer is in the CS, no other process may be in the CS at that time. However, since readers do not modify the buffer, the exclusion requirement is relaxed for readers: any number of readers are allowed to be in the CS simultaneously. A reader-writer exclusion algorithm takes advantage of this relaxation in the exclusion requirement, besides satisfying several other desirable properties, such as concurrent entering, first-in-first-enabled among the readers,

first-come-first-served among the writers, and bounded-exit. These properties are defined later in Section 2.

When readers and writers compete for the CS, the algorithm has three natural choices: (i) give higher priority to readers, (ii) give higher priority to writers, or (iii) neither class has a higher priority and no reader or writer starves. In this paper we consider only the last (starvation-free) case.

1.2 Remote Memory Reference (RMR) Complexity

In an algorithm that runs on a multiprocessor, if a process p accesses a shared variable that resides at p's local memory module, the access will be fast, but if p accesses a remote shared variable, the access can be extremely slow (because of the delay in gaining exclusive access to the interconnection bus and the high latency of the bus). Research in the last two decades has therefore been driven by the goal to minimize the number of *remote memory references (RMRs)* (see the survey [1]). Most mutual exclusion algorithms are designed for two shared memory models [1]: *Distributed Shared Memory* (DSM) model [16] and *Cache-Coherent* (CC) model [21]. In DSM model, each process has its local shared memory module which can be accessed by all processes. A reference to a shared variable X is considered remote if X is at a memory module of a different process. In CC model, the shared memory modules are remote from all processes. However, each process has its own cache which can only be accessed by the process itself. When a process p reads a shared variable X in the shared memory, p will store X in its cache. If p wants to read X again, it can read the copy of X in its cache instead of accessing the remote shared memory as long as X has not been updated by other processes since p's previous read of X. If X has not been updated, the copy of X in p's cache will indicate itself as invalid. Thus, when p's wants to read X, it

has to access the shared memory. Hence, a reference to X by p is considered remote in CC model if X is not in p's cache.

The goal of minimizing the number of RMRs implies that algorithms should be designed to achieve local spinning, i.e., processes do not make any remote references in busywait loops. The ideal goal is to design locking algorithms whose *RMR complexity*–the worst case number of remote memory references made by a process to enter and exit the CS once–is a constant, independent of the number of processes executing the algorithm. This goal was achieved for mutual exclusion over twenty years ago—Anderson's algorithm achieves constant RMR complexity for CC machines [2], and Mellor-Crummey and Scott's algorithm achieves constant RMR complexity for both CC and DSM machines [17].

In contrast, for the reader-writer problem, constant RMR complexity is achievable for CC machines [3, 4], but is provably impossible for DSM machines: Danek and Hadzilacos' lower bound proof for 2-Session Group Mutual Exclusion implies that a sublinear RMR complexity algorithm satisfying concurrent entering is impossible for the reader-writer exclusion problem [10].

1.3 Abortability

In certain applications, such as real-time operating systems and databases, indefinite waiting can cause a process to miss an important deadline [20]. Therefore, there has been a lot of research on the design of exclusion algorithms that provide an extra feature—the *Abort Section*—that a busywaiting process in the Try Section can execute if it wishes to quit the protocol [20, 19, 11, 14]. Since a process executes the Abort Section only when it cannot afford to wait any further, it is imperative that this section of code be *wait-free*, i.e., a process completes the Abort Section in a bounded number of its own steps, regardless of

how its steps interleave with the steps of other processes.

For the mutual exclusion problem, efficient abortable algorithms are known: Jayanti's algorithm has $O(\log n)$ RMR complexity, where n is the number of processes [11]. Lee's algorithm has the same worst-case complexity, but achieves $O(1)$ complexity for the case where no process aborts [14]. For the reader-writer exclusion problem, Zheng designed an abortable algorithm of $O(n)$ RMR complexity, but this algorithm applies only for the reader-priority case [22]. To the best of our knowledge, there is no abortable algorithm for the starvation-free case, which is the focus of this paper.

1.4 The Main Result

We investigate the hardness of the abortable reader-writer exclusion problem relative to the abortable mutual exclusion problem. Let $me(n)$ and $rw(n)$ denote the worst case RMR complexities of the abortable, starvation-free, mutual exclusion problem and the abortable, starvation-free reader-writer exclusion problem, respectively. Since mutual exclusion is a special case of reader-writer exclusion where all processes act as writers, it follows that $me(n) \leq rw(n)$, i.e., $me(n) = O(rw(n))$. Is the converse true? To our surprise, we found the answer is yes. Specifically, we present a constant RMR complexity transformation that converts any abortable, starvation-free mutual exclusion algorithm into an abortable, starvation-free reader-writer exclusion algorithm. This establishes that $rw(n) \leq me(n) + O(1)$, and thus $rw(n) = O(me(n))$.

Our result has two significant implications:

- It establishes that $rw(n) = \Theta(me(n))$, i.e., abortable, starvation-free, reader-writer exclusion is exactly as hard as abortable, starvation-free, mutual exclusion.

- Our transformation, when applied to the $O(\log n)$ abortable mutual exclusion algorithm, gives rise to an abortable, starvation-free reader-writer exclusion algorithm of $O(\log n)$ RMR complexity. To the best of our knowledge, this is the first abortable, starvation-free reader-writer exclusion algorithm.

1.5 How the Transformation is Structured

Our transformation is presented in two steps. First, in Section 3, we design an abortable reader-writer algorithm $\mathcal{A}$ that supports only a single writer. Then, in Section 4, we show how to combine $\mathcal{A}$ with an abortable mutual exclusion lock $\mathcal{M}$ to obtain an abortable reader-writer algorithm that supports an arbitrary number of writers (and readers). The design of $\mathcal{A}$ in the first step constitutes the intellectual contribution of this paper. (The second step uses the simple idea that multiple writers compete for the lock $\mathcal{M}$ and the successful one proceeds to execute the single-writer algorithm $\mathcal{A}$.)

2 The Abortable Reader-Writer Exclusion Problem

In this section we provide a clear statement of the abortable reader-writer exclusion problem.

Each process has five sections of code—Remainder, Try, Critical, Exit, and Abort Sections. A process executes its code in phases. In each phase, the process does one of two things: (1) starts in the Remainder Section; then executes the Try Section, the Critical Section (CS), and the Exit Section (in that order); and then goes back to the Remainder Section, or (2) starts in the Remainder Section; then executes the Try Section, possibly partially; then executes the Abort Section; and then goes back to the Remainder Section.

The Try Section is divided into a *doorway*, followed by a *waiting room* [13]. It is required

that the doorway be wait-free, i.e., each process completes the doorway in a bounded number of its steps, regardless of how its steps interleave with the steps of other processes.

We say a reader r in the Try Section is *enabled* if r will enter the CS in a bounded number of its own steps, regardless of how its steps interleave with the steps of other processes.

The *reader-writer exclusion problem* is to design the Try, Exit, and Abort Sections of code for each process so that the following properties hold in all runs:

- (P1) Reader-Writer Exclusion: If a writer is in the CS, then no other process is in the CS at that time.

- (P2) Bounded Abort: Each process completes the Abort Section in a bounded number of its steps, regardless of how its steps interleave with the steps of other processes.

- (P3) Concurrent Entering: Since readers don't conflict with each other, it is desired that they do not obstruct each other from entering the CS. More specifically, if all writers are in the Remainder Section and will remain there, then every reader in the Try Section enters the CS in a bounded number of its own steps [9, 12].

- (P4) FIFE among readers: Unlike writers that may only access the CS one at a time, any number of readers can cohabit the CS. Consequently, if a reader r completes the doorway before another reader r' enters the doorway, there is no reason to delay the entry of r' into the CS for the sake of r. We use the *First-In-First-Enabled* (FIFE) property, first defined by Fischer et al. for the k-exclusion problem [8], to define fairness among readers, as follows: If a reader r in the Try Section completes the doorway before another reader r' enters the doorway, and r' subsequently enters the CS, then one of the following three conditions holds: (i) r enters the CS before r'

enters the CS, or (ii) r begins executing the Abort Section before r' enters the CS, or (iii) r is enabled to enter the CS when r' enters the CS.

- (P5) Starvation-Freedom: When readers and writers compete for the CS, the algorithm has three natural choices: (i) give higher priority to readers, (ii) give higher priority to writers, or (iii) treat both classes of processes fairly. In this paper we consider only the third case and require the *starvation-freedom* property: if a process in the Try Section does not abort, it will eventually enter the CS, under the assumption that no process stops taking steps in the Try, Exit, or Abort Sections and no process stays in the CS forever.

- (P6) Bounded Exit: Every process completes the Exit Section in a bounded number of its own steps.

- (P7) FCFS among writers: We use the *First-Come-First-Served (FCFS)* property, first defined by Lamport for the mutual exclusion problem [13], to define fairness among writers, as follows: If a writer w completes the doorway before a writer w' enters the doorway and w does not abort, then w' does not enter the CS before w.

3 Single-Writer Multi-Reader Algorithm

3.1 The Algorithm

Now we present our abortable starvation-free algorithm that works for single writer and multiple readers. The algorithm employs shared variables that support read/write and fetch&add (F&A) operations, where fetch&add operation is defined as follows:

- $F\&A(X, a)$ is a fetch&add operation on variable X and it makes $X = x + a$ and

returns $(x + a)$, where x is the previous value of X.[1]

Figure 1 shows our abortable single-writer multi-reader algorithm. The subroutine R-Abort is what readers execute when they abort from Line 2, while W-Abort-L7 and W-Abort-L11 are what the writer executes when it aborts from Line 7 and Line 11 respectively. W-Abort-L7 is empty, meaning the writer can quit immediately if it aborts from Line 7. It is worth noting that this algorithm works for any number of readers

---

G: integer
$Flag$: single bit
X: pair of integers, where the first component only consists of one bit
Y: integer
Initially, $X = [0, 0]$, $Y = 0$, $Flag = 0$, $G = 2$

Reader
1. $[g, -] \leftarrow F\&A(X, [0, 1])$
2. wait till $G \geq 2 \vee G = g$
3. CS
   do R-Exit

R-Exit
4. $[g', -] \leftarrow F\&A(X, [0, -1])$
   if $g' \neq g$
5. $\quad a \leftarrow F\&A(Y, 1)$
   $\quad$ if $(a = 0)$
6. $\quad\quad F\&A(Flag, 1)$

R-Abort
do R-Exit

Writer
7. wait till $Flag = 0$
8. $F\&A(G, -2)$
9. $[s, r] \leftarrow F\&A(X, [1, 0])$
10. $c \leftarrow F\&A(Y, -r)$
    if $c \neq 0$
11. $\quad$ wait till $Flag = 1$
12. CS
    do R-Exit

W-Exit
13. $G \leftarrow s + 2$
    if $(c \neq 0)$
14. $\quad F\&A(Flag, 1)$

W-Abort-L7
do nothing

W-Abort-L11
do W-Exit

Figure 1: Abortable Single-Writer Multi-Reader Algorithm

To fully understand the algorithm, we first need to know the purposes of the shared variables the algorithm uses.

[1]People usually assume that fetch&add returns the previous value of the variable. But for convenience, we assume in this paper that it returns the new value: we assume the operation returns $(x + a)$, not x.

- G: Intuitively, G is an entrance that has two gates, Gate 0 and Gate 1, through which readers enter the CS. When the writer is requesting to enter the CS or occupying the CS, either $G = 0$ or $G = 1$, indicating either only Gate 0 or only Gate 1 is open. This "gates" idea was proposed by Bhatt and Jayanti [3]. Here, in order to support aborting, we do the following modification: if the writer is not requesting to or occupying the CS, we let $G \geq 2$, indicating both gates are open.

- X: X has two components. Let us call them $X.1$ and $X.2$ from left to right. $X.1$ is a single bit read by readers to figure out which gate they should pass. $X.2$ is a counter for the number of readers that are requesting to enter or currently in the CS. When the writer wants to access the CS, it reads $X.2$ at Line 9 to know the number of readers that are in the CS or about to enter the CS. The writer then waits until all such readers have exited the CS.

- Y: Y is what the writer uses to communicate with readers about how many readers it needs to wait for. As we mentioned above, the writer gets from $X.2$ the number of readers that are in the CS or about to enter the CS. Then the writer writes this information into Y by subtracting Y by that number at Line 10. We will explain later that only the readers that the writer needs to wait for will find $g' \neq g$ at Line 4 and therefore go to Line 5 to increment Y. Thus, when the last of these readers executes Line 5, it will find $a = 0$, i.e., $Y = 0$, and then try to wake up the writer by executing Line 6.

- $Flag$: $Flag$ is a single bit indicating what the writer should do. As we mentioned before, when the writer is waiting for some readers to leave the CS at Line 11, the last of such readers will go to Line 6 to increment $Flag$. Thus, when the writer finds

$Flag = 1$, it knows that it can now enter the CS. When the writer exits, it flips $Flag$ back to 0 by executing Line 14. On the other hand, If the writer does not want to wait any longer at Line 11, it can abort and then execute Line 14 to increment $Flag$. When the writer enters the Try Section again, some slow readers may not have exited yet. In this case, the writer will find $Flag = 1$ at Line 7 because of the previous increment at Line 14. When the last of such readers finally leaves, it increments $Flag$ at Line 6 so as to set $Flag = 0$. Now the writer will find $Flag = 0$ and hence can move on to request its access to the CS.

With the above description of the shared variables, we can now explain the details of the algorithm. We begin by describing how readers get the permission to enter the CS. When a reader enters the Try Section, it reads the first component of X, $X.1$, that indicates which gate the reader will pass through. If the writer is now in the Remainder Section, we have $G = X.1 + 2$ and hence $G \geq 2$. Therefore, the reader knows it is allowed to enter the CS. If the writer has entered the Try Section and just executed Line 8, we have $G = X.1$. Since $X.1$ is unchanged, the reader has $g = X.1 = G$ and can therefore enter the CS. After the writer executed Line 9 to flip $X.1$ by incrementing it by 1, we have $X.1 \neq G$. If a reader now comes in the Try Section and executes Line 1, it will find $G < 2$ and $G \neq g$ and hence need to wait at Line 2. However, any reader that executed Line 1 before the writer executed Line 9 still has $G = g$, since $g = X.1 = G$ held at the time it executed Line 1 and G is now still having the same value. Therefore, Line 9 becomes the critical point for synchronization: if a reader comes in the Try Section before the writer executes Line 10, it is *enabled* to enter the CS; otherwise, it has to let the writer enter the CS first. When the writer exits either after it left the CS or after it decided to abort, it will execute Line 13. After that moment,

$G = X.1 + 2$ (since the writer stored $X.1$'s value in s), and hence all the waiting readers are now enabled because $G \geq 2$. There is only one risk for readers: when the writer is in the Remainder, a reader R executes Line 1 and then falls asleep before executing Line 2. The writer now enters the Try Section and aborts, making $G \geq 2$ but $G \neq R.g + 2$. Then the writer enters the Try Section again, executing Line 8 to make $G < 2$ and $G \neq R.g$. If R now wakes up, it will be blocked from entering the CS, which we do not expect to happen. However, this situation will not happen: the second time the writer enters the Try Section, it will find $Flag = 1$ at Line 7 and hence it cannot execute Line 8 to cause that risk. We will explain the reason later.

Let us now explain how the writer gets access to the CS. Initially, $Flag = 0$. Therefore, the writer finds $Flag = 0$ the first time it enters the Try section. Then, it executes Line 8 and Line 9, claiming its request to access the CS. As we mentioned earlier, any readers that enter the Try Section after the writer's request will be blocked from getting into the CS. At the same time, the writer gets from $X.2$ the number of enabled readers that have not exited yet. Then, at Line 10, the writer subtracts Y by the number of such readers, r. Note that all the readers that entered the Try Section before the writer requested the CS have $g \neq X.1$ now because of the change on $X.1$ at Line 9. Therefore, if these readers exit or abort (these two subroutines are actually the same), they will find $g' = X.1 \neq g$ at Line 4 and hence execute Line 5 to increment Y. On the other hand, if any blocked reader waiting at Line 2 aborts (or finally enters and then leaves the CS after the writer has left), it will find $g' = g$ because it executed Line 1 after the writer made the change on Y at Line 9. Therefore, it will leave without touching Y. Hence, when the writer gets the value c at Line 10 after subtracting Y by r, it knows that there are exactly $-c$ enabled readers

it has to wait for. If $c \neq 0$, the writer will wait until the last such reader increments Y to 0 and sets $Flag$ to 1. When the writer exits, it sets $Flag$ back to 0 so that it will find $Flag = 0$ next time it enters the Try Section. If the writer decides to abort from Line 11, it also increments $Flag$ by 1. Thus, if the last of those enabled readers has not exited or aborted yet, $Flag = 1$. Therefore, when the writer enters the Try Section again, it has to wait at Line 7 until $Flag = 0$, meaning the last reader has gone. On the other hand, if the writer finds $c = 0$ after executing Line 10, it knows that all enabled readers have executed Line 5 before it executes Line 10. Hence, the writer can enter the CS immediately. An important observation is that, if a reader executed Line 5 before the writer executes Line 10, it incremented Y to a positive value and hence found $a > 0$. Therefore, in the case where writer has $c = 0$, no reader can go to Line 6 to change $Flag$. Hence, the writer can just leave $Flag$ unchanged by not executing Line 14 and it will find $Flag = 0$ next time it enters the Try Section. Now consider that the writer comes into the Try Section again. If it finds $Flag = 1$ at Line 7, it knows that some enabled readers with $g' \neq g$ (or with $g \neq X.1$ at Line 2–4) have not left. Hence, the writer waits until they all leave and $Flag$ is therefore set back to 0. If it aborts from Line 7, it does nothing. But next time it enters the Try Section, it still needs to wait there until $Flag = 0$. It is easy to figure out that, if any reader enters and leaves during the period when the writer is in the Remainder Section and at Line 7, it will find $g = g'$ at Line 4. Hence, such readers cannot change Y or $Flag$. Therefore, the writer does not need to worry about the risk that some reader updates Y or $Flag$ improperly. After the writer passes Line 7, it can then request the CS by executing the Try Section, the same as what it did for the first time.

3.2 Correctness of the Algorithm

3.2.1 Invariants of the Algorithm

Our proof is invariant-based. In Figure 2, we present the invariant $\mathcal{I}$ that our single-writer multi-reader algorithm satisfies.

In the following, we prove $\mathcal{I}$ always holds by showing that $\mathcal{I}$ holds in the initial configuration, and that, if $\mathcal{I}$ holds in any reachable configuration $\mathcal{C}$, it still hold after any process takes a step.

- **Lemma 3.1** *$\mathcal{I}$ holds in the initial configuration.*

  Proof of Lemma 3.1: In the initial configuration, the writer W is at Line 7 and all readers are at Line 1. Hence, we need to prove that $\mathcal{I}_G$ and $\mathcal{I}_7$ hold. By simply checking the initially values of the shared variables, we can prove all items of $\mathcal{I}_G$ and $\mathcal{I}_7$ hold in the initial configuration. □

- **Lemma 3.2** *If $\mathcal{I}$ holds in configuration $\mathcal{C}$, then it also holds in $\mathcal{C}.s$ where s is an arbitrary step taken by an arbitrary process.*

  Proof of Lemma 3.2: First, we prove $\mathcal{I}_G$ holds in any reachable configuration. Item 1 of $\mathcal{I}_G$ is true, since, for any reader R, $R.g$'s value is from $X.1$ at Line 1 which consists of one bit. Since any read attempt increments $X.2$ by 1 once at Line 1 and then decrements it by 1 once at Line 4, $X.2$ must be equal to the number of read attempts that have executed Line 1 and have not done Line 4 yet. Hence, Item 2 of $\mathcal{I}_G$ is true in any reachable configuration. Therefore $\mathcal{I}_G$ always holds.

  In the following, we prove in Claim 3.3–Claim 3.10 that if W is at Line i in configuration $\mathcal{C}$ then $\mathcal{I}$ holds in $\mathcal{C}.s$, for any $i \in \{7, 8, 9, 10, 11, 12, 13, 14\}$. Hence, we

- Definitions:
  1. R and W denote a reader and the writer, respectively.
  2. $\mathcal{I}_i$ is a collection of predicates that are true when the writer's program counter is at Line i.
  3. $PC_R = i$ states that reader R's program counter is at Line i.
  4. $X.1$ denotes the second field of X. Similar definitions for $X.2$, $Y.1$, etc.

- $\mathcal{I}_G$ (global invariant):
  1. $\forall R, PC_R \in \{2,3,4,5,6\} \Longrightarrow R.g = 1 \vee R.g = 0$
  2. $X.2 = |\{R | PC_R \in \{2,3,4\}\}|$

- $\mathcal{I}_7$:
  1. $G = X.1 + 2$
  2. $Y = -|\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}|$
  3. $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$
  4. $Y \neq 0 \Longrightarrow (Flag = 1 \wedge |\{R | PC_R = 6\}| = 0)$
  5. $(Flag = 1 \wedge Y = 0) \Longrightarrow |\{R | PC_R = 6\}| = 1$
  6. $Flag = 0 \Longrightarrow Y = 0 \wedge |\{R | PC_R = 6\}| = 0$

- $\mathcal{I}_8$:
  1. $G = X.1 + 2$
  2. $Y = 0$
  3. $|\{R | PC_R \in \{2,3,4\} \wedge R.g \neq X.1\}| = 0$
  4. $|\{R | PC_R \in \{5,6\}\}| = 0$
  5. $Flag = 0$

- $\mathcal{I}_9$:
  1. $G = X.1$
  2. $Y = 0$
  3. $|\{R | PC_R \in \{2,3,4\} \wedge R.g \neq X.1\}| = 0$
  4. $|\{R | PC_R \in \{5,6\}\}| = 0$
  5. $Flag = 0$

- $\mathcal{I}_{10}$:
  1. $G = 1 - X.1$
  2. $Y = W.r - |\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}| \geq 0$
  3. $\forall R, PC_R = 3 \Longrightarrow R.g \neq X.1$
  4. $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$
  5. $|\{R | PC_R \in 6\}| = 0$
  6. $Flag = 0$
  7. $W.s = X.1$

- $\mathcal{I}_{11}$:
  1. $G = 1 - X.1$
  2. $Y = -|\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}| \leq 0$
  3. $\forall R, PC_R = 3 \Longrightarrow R.g \neq X.1$
  4. $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$
  5. $Y \neq 0 \Longrightarrow Flag = 0 \wedge |\{R | PC_R = 6\}| = 0$
  6. $Flag = 0 \wedge Y = 0 \Longrightarrow |\{R | PC_R = 6\}| = 1$
  7. $Flag = 1 \Longrightarrow Y = 0 \wedge |\{R | PC_R = 6\}| = 0$
  8. $W.s = X.1$
  9. $W.c \neq 0$

- $\mathcal{I}_{12}$:
  1. $G = 1 - X.1$
  2. $Y = |\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}| = 0$
  3. $|\{R | PC_R \in \{3,5,6\}\}| = 0$
  4. $Flag = 0 \Longleftrightarrow W.c = 0$
  5. $W.s = X.1$

- $\mathcal{I}_{13}$:
  1. $G = 1 - X.1$
  2. $Y = -|\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}| \leq 0$
  3. $\forall R, PC_R = 3 \Longrightarrow R.g \neq X.1$
  4. $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$
  5. $W.c = 0 \Longrightarrow Y = 0$
  6. $W.c = 0 \Longrightarrow |\{R | PC_R = 6\}| = 0$
  7. $W.c = 0 \Longrightarrow Flag = 0$
  8. $Y \neq 0 \Longrightarrow (Flag = 0 \wedge |\{R | PC_R = 6\}| = 0)$
  9. $(W.c \neq 0 \wedge Flag = 0 \wedge Y = 0) \Longrightarrow |\{R | PC_R = 6\}| = 1$
  10. $(W.c \neq 0 \wedge Flag = 1) \Longrightarrow (Y = 0 \wedge |\{R | PC_R = 6\}| = 0)$
  11. $W.s = X.1$

- $\mathcal{I}_{14}$:
  1. $G = X.1 + 2$
  2. $Y = -|\{R | PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}| \leq 0$
  3. $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$
  4. $Y \neq 0 \Longrightarrow (Flag = 0 \wedge |\{R | PC_R = 6\}| = 0)$
  5. $(Flag = 0 \wedge Y = 0) \Longrightarrow |\{R | PC_R = 6\}| = 1$
  6. $Flag = 1 \Longrightarrow (Y = 0 \wedge |\{R | PC_R = 6\}| = 0)$

Figure 2: Invariant $\mathcal{I}$ of the Abortable Single-Writer Multi-Reader Algorithm

can conclude that $\mathcal{I}$ holds in $\mathcal{C}.s$, where s is an arbitrary step taken by an arbitrary process. □

Claim 3.3 *If $PC_W = 7$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.3:

– If s is a step taken by a reader R, then $PC_W = 7$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Since R cannot change G, Item 1 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Since Item 3 of $\mathcal{I}_7$ holds in $\mathcal{C}$, it still holds if s does not make any effect on it. Item 3 of $\mathcal{I}_7$ can only be affected if R executes Line 4 and goes to Line 5. To do so, however, R has to find $R.g' \neq R.g$ at Line 4, where $R.g' = X.1$. This implies that $R.g \neq X.1$ and therefore Item 3 of $\mathcal{I}_7$ still holds in $\mathcal{C}.s$.

Since Item 2 of $\mathcal{I}_7$ holds, it still holds if s does not make any effect on it. Item 2 of $\mathcal{I}_7$ can only be affected when **(a)** R is at Line 1 in $\mathcal{C}$, or **(b)** R is at Line 5 with $R.g \neq X.1$ (since Item 3 of $\mathcal{I}_7$ holds in $\mathcal{C}$, the possibility of $PC_R = 5$ with $R.g = X.1$ in $\mathcal{C}$ has been ruled out). In case **(a)**, R will have $PC_R = 2$ and $R.g = X.1$ after R takes a step. Hence, Item 2 $\mathcal{I}_7$ holds in this case. In case **(b)**, R increments Y by 1 and will go to Line 6 or Line 1 after taking a step. Thus, both sides of the equation in Item 2 increase by 1 and hence remain equal. Hence, Item 2 $\mathcal{I}_7$ holds in this case. Therefore Item 2 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Item 4 of $\mathcal{I}_7$ can only be affected when $Y \neq 0$ and $PC_R = 5$ in $\mathcal{C}$ (all other possibilities are ruled out, since $\mathcal{I}_7$ holds in $\mathcal{C}$). To affect Item 4 in this case, R has to go to Line 6 after taking a step. To achieve this, R has to find $R.a = 0$,

which requires $Y = 0$ after s is taken. Thus, Item 4 holds unconditionally. Therefore Item 4 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Item 5 of $\mathcal{I}_7$ can only be affected when **(a)** $PC_R = 5$ $\mathcal{I}_7$ in $\mathcal{C}$, or **(b)** $PC_R = 6$ in $\mathcal{C}$. In case **(a)**, R has to get into Line 6 in $\mathcal{C}.s$, which requires $Y = -1 \neq 0$ before R takes a step. Thus, Item 4 of $\mathcal{I}_7$ implies that $|\{R|PC_R = 6\}| = 0$ holds in $\mathcal{C}$. Therefore, we have $|\{R|PC_R = 6\}| = 1$ after one step and hence Item 5 holds. In case **(b)**, we can conclude that R is the only reader at Line 6 since Items 4–6 of $\mathcal{I}_7$ imply $|\{R|PC_R = 6\}| \leq 1$ in $\mathcal{C}$. Moreover, we can figure out that the only possibility is that $Flag = 1$ and $Y = 0$ in $\mathcal{C}$. Thus, after R takes a step, $Flag = 0$ and Item 5 is true unconditionally. Therefore Item 5 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Since $\mathcal{I}_7$ holds in $\mathcal{C}$, to affect Item 6 in $\mathcal{C}.s$, we need to have **(a)** $Flag = 0$ and $PC_R = 5$, or **(b)** $Flag = 1$, $Y \neq 0$ and $PC_R = 6$ in $\mathcal{C}$. However, Items 2,3, and 6 of $\mathcal{I}_7$ in $\mathcal{C}$ rule out the possibility of **(a)** while Item 4 rules out the possibility of **(b)**. Therefore Item 6 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Hence, if s is a step taken by a reader R, $\mathcal{I}$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W and $Flag = 1$ in $\mathcal{C}$, or if W aborts in $\mathcal{C}$, then $PC_W = 7$ in $\mathcal{C}.s$. Since s does not make any change, $\mathcal{I}$ still holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W and $Flag = 0$ in $\mathcal{C}$, then $PC_W = 8$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_8$ holds in $\mathcal{C}.s$.

Note that executing Line 7 does not change any shared variables. Since $Flag = 0$ and $\mathcal{I}_7$ holds in $\mathcal{C}$, we know $Y = 0$ and $|\{R|PC_R = 6\}|$. Combining these facts and Items 1–3 of $\mathcal{I}_7$, we can figure out that all Items 1–5 of $\mathcal{I}_8$ hold in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.4 *If $PC_W = 8$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.4:

- If s is a step taken by a reader R, then $PC_W = 8$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_8$ holds in $\mathcal{C}.s$.

  By the same argument in the proof of Claim 3.3, we can prove Item 1 of $\mathcal{I}_8$ holds in $\mathcal{C}.s$.

  Since $\mathcal{I}_8$ holds in $\mathcal{C}$, we know R can be at Line 5 or Line 6. Thus, R cannot change Y or $Flag$. Hence, Items 2 and 5 of $\mathcal{I}_8$ still hold in $\mathcal{C}.s$.

  If $PC_R = 1$ in $\mathcal{C}$, then $R.g = X.1$. Combining this and the fact that Item 3 of $\mathcal{I}_8$ holds in $\mathcal{C}$, we can conclude that Item 3 still $\mathcal{I}_8$ holds in $\mathcal{C}.s$. If s is a step taken by a reader R, then $PC_W = 8$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_8$ holds in $\mathcal{C}.s$. According to Item 3 of $\mathcal{I}_8$, if $PC_R = 4$, then $R.g = X.1$. Thus, R cannot be at Line 5 in $\mathcal{C}.s$. This and the fact that Item 4 of $\mathcal{I}_8$ holds in $\mathcal{C}$ together indicate that Item 4 holds in $\mathcal{C}.s$.

  Therefore, If s is a step taken by a reader, $\mathcal{I}$ holds in $\mathcal{C}.s$.

- If s is a step taken by the Writer W, then $PC_W = 9$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_9$ holds in $\mathcal{C}.s$. Since $\mathcal{I}_8$ holds in $\mathcal{C}$ and executing Line 8 only decrements G by 2, we can figure out that $\mathcal{I}_9$ holds in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.5 *If $PC_W = 9$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.5:

– If s is a step taken by a reader R, then $PC_W = 9$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_9$ holds in $\mathcal{C}.s$.

  By the same argument in the proof of Claim 3.4, we can prove that $\mathcal{I}_9$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W, then $PC_W = 10$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

  Since $G = X.1$ in $\mathcal{C}$, we have $G = 1 - X.1$ after W executes Line 9, i.e., Item 1 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

  Since Item 3 of $\mathcal{I}_9$ holds in $\mathcal{C}$ and executing Line 9 flips $X.1$, we have $|\{R|PC_R \in \{2,3,4\} \wedge R.g = X.1\}| = 0$ in $\mathcal{C}.s$. Thus, Item 3 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

  Since W stores the value of X in $W.s$ and $W.r$, Item 7 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$. What's more, since $|\{R|PC_R \in \{2,3,4\} \wedge R.g \neq X.1\}| = |\{R|PC_R \in \{2,3,4\}\}| = X.1$ and $Y = 0$ in $\mathcal{C}$, we have $Y = W.r - |\{R|PC_R \in \{2,3,4\} \wedge R.g \neq X.1\}| = 0$, i.e., Item 2 of $\mathcal{I}_{10}$ in $\mathcal{C}.s$.

  Since Items 4 and 5 of $\mathcal{I}_9$ hold in $\mathcal{C}$, we know Items 4–6 of $\mathcal{I}_{10}$ hold in $\mathcal{C}.s$.

  Therefore, if s is a step taken by the Writer W, $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.6 *If $PC_W = 10$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.6:

– If s is a step taken by a reader R, then $PC_W = 10$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

Since Items 1 and 7 of $\mathcal{I}_{10}$ hold in $\mathcal{C}$ and they cannot be affected by a step taken by a reader, they still hold in $\mathcal{C}.s$.

Since Item 3 $\mathcal{I}_{10}$ holds in $\mathcal{C}$, to violate Item 3 in $\mathcal{C}.s$, R has to execute line 2 with $R.g \neq X.1$ and $R.g = G$ so that R can go to line 3 with $R.g \neq X.1$. However, Item 1 of $\mathcal{I}_{10}$ rules out this possibility. Hence, Item 3 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$. Likewise, we can prove Item 4 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

If R is at Line 2, Line 3, or Line 6 and it takes a step, Item 2 of $\mathcal{I}_{10}$ won't be affected and will still hold. If R executes Line 1, it will have $R.g = X.1$ and hence will not affect Item 2. If R with $R.g \neq X.1$ executes Line 4, it will find $R.g \neq R.g'$ and then go to Line 5, which will not violate Item 5. If $R.g = X.1$, it will not affect Item 5. If R executes Line 5, both sides of the equation in Item 5 will increase by 1, and hence Item still holds. Therefore, Item 5 must hold in $\mathcal{C}.s$.

According to Item 2 and Item 5 of $\mathcal{I}_{10}$, $Y \geq 0$ holds in $\mathcal{C}$ and no reader is at Line 5. Thus, To violate Item 5 of $\mathcal{I}_{10}$ in $\mathcal{C}.s$, R has to execute Line 6 with $Y = -1$ in $\mathcal{C}$, so that R can make $Y = R.a = 0$ to go to Line 6. However, we know $Y \geq 0$ in $\mathcal{C}$ and hence this is impossible. Hence, Item 5 of $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$.

Since Item 5 of $\mathcal{I}_{10}$ holds in $\mathcal{C}$, R cannot change $Flag$. Hence, Item 6 $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$. Therefore, $\mathcal{I}_{10}$ holds in $\mathcal{C}.s$

– If s is a step taken by the Writer W and $Y - W.r = 0$ in $\mathcal{C}$, then $W.c = 0$ and hence $PC_W = 12$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

Since , by $\mathcal{I}_{10}$, Items 1 and 5 of $\mathcal{I}_{12}$ hold in $\mathcal{C}$ and they remain unchanged after s is taken, they still hold in $\mathcal{C}.s$. Since $Flag = 0$ holds in $\mathcal{C}$ and we have $W.c$, Item

4 of $\mathcal{I}_{12}$ also holds in $\mathcal{C}.s$.

Since we have $Y - W.r = 0$ and Item 2 of $\mathcal{I}_{10}$ in $\mathcal{C}$, we know Item 2 of $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$. In addition, Items 3–5 of $\mathcal{I}_{10}$ hold in $\mathcal{C}$. Hence, we can conclude Item 3 of $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

Therefore, $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W and $Y - W.r \neq 0$ in $\mathcal{C}$, then $W.c \neq 0$ and hence $PC_W = 11$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

Like the proof above, we can prove that Items 1 and 8 of $\mathcal{I}_{11}$ hold in $\mathcal{C}.s$. Since we assume $W.c \neq 0$ in this case, Item 9 of $\mathcal{I}_{11}$ also holds.

Since Item 2 of $\mathcal{I}_{10}$ holds in $\mathcal{C}$, and the execution of Line 10 subtracts Y by $W.r$, we know Item 2 of $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

Since Items 3 and 4 $\mathcal{I}_{11}$ cannot be affected by the execution of Line 10 and they hold in $\mathcal{C}$, they still hold in $\mathcal{C}.s$.

Since $Flag = 0$ in $\mathcal{C}$, $Flag = 0$ still holds in $\mathcal{C}.s$. We also assume in this case that $W.c = Y \neq 0$. Hence, Items 6 and 7 of $\mathcal{I}_{11}$ hold unconditionally in $\mathcal{C}.s$. Moreover, since Item 5 of $\mathcal{I}_{10}$ holds in $\mathcal{C}$, Item 5 of $\mathcal{I}_{11}$ will hold in $\mathcal{C}.s$.

Therefore, $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.7 *If $PC_W = 11$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.7:

– If s is a step taken by a reader R, then $PC_W = 11$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

Similar to the proof of Claim 3.6, we can prove Items 1–4 and 8 of $\mathcal{I}_{11}$ hold in $\mathcal{C}.s$. Since $W.c$ is W's local variable, Item 9 of $\mathcal{I}_{11}$ also holds in $\mathcal{C}.s$.

To violate Items 5 of $\mathcal{I}_{11}$, R has two ways: **(a)** R executes Line 5 when $Y \neq -1$ and then go to Line 6, or **(b)** R executes Line 5 when $Y = 0$ and another reader is already at Line 6. Case **(a)** cannot happen, since R will find $R.a = 0$ and then cannot go to Line 6. Case **(b)** cannot happen either, since, by Items 2 and 4 of $\mathcal{I}_{11}$, R cannot be at Line 5 in $\mathcal{C}$. Hence, Item 5 of $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

To violate Item 6 of $\mathcal{I}_{11}$, R has two ways: **(a)** R executes Line 6 when $Flag = 1$ and $Y = 0$, or **(b)** R executes Line 5 when $Y = -1$ and another reader is already at Line 6. However, case **(a)** cannot happen, since, by Item 7 of $\mathcal{I}_{11}$, R cannot be at Line 6 in $\mathcal{C}$. Case **(b)** cannot happen either, since, by Items 5 of $\mathcal{I}_{11}$, R cannot be at Line 5 in $\mathcal{C}$. Hence, Item 5 of $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

To violate Item 7 of $\mathcal{I}_{11}$, R has two ways: **(a)** R executes Line 5 when $Flag = 1$ and $Y = -1$, or **(b)** R executes Line 6 when $Flag = 0$ and another reader is already at Line 6. However, case **(a)** cannot happen, since, by Item 5 of $\mathcal{I}_{11}$, we know $Flag = 0$ in $\mathcal{C}$. Case **(b)** cannot happen either, since, by Items 5–7 of $\mathcal{I}_{11}$, we know there is at most one reader in $\mathcal{C}$. Hence, Item 5 of $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

Therefore, $\mathcal{I}_{11}$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W and $Flag = 0$ in $\mathcal{C}$, then $PC_W = 11$ in $\mathcal{C}.s$.

Since the execution of Line 11 doesn't change anything, $\mathcal{I}_{11}$ still holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W and $Flag = 1$ in $\mathcal{C}$, then $PC_W = 12$ in $\mathcal{C}.s$.

Therefore, we need to prove $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

Since $Flag = 1$ in this case and Items 1, 2, 3, 4, 7, 8, and 9 $\mathcal{I}_{11}$ hold in $\mathcal{C}$, we can

conclude that all the Items 1-5 of $\mathcal{I}_{12}$ hold in $\mathcal{C}.s$.

- If W aborts in $\mathcal{C}$, then $PC_W = 13$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{13}$ holds in $\mathcal{C}.s$.

  Since Items 1, 2, 3, 4, 5, 6, 8, and 9 of $\mathcal{I}_{11}$ hold in $\mathcal{C}$, we can figure out that Items 1, 2, 3, 4, 8, 9, 10, and 11 of $\mathcal{I}_{13}$ hold in $\mathcal{C}.s$. Moreover, Items 5–7 of $\mathcal{I}_{13}$ hold unconditionally in $\mathcal{C}.s$. Therefore, $\mathcal{I}_{13}$ holds in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.8 *If $PC_W = 12$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.8:

- If s is a step taken by a reader R, then $PC_W = 12$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

  Similar to the proof of Claim 3.6, we can prove that Items 1, 2, and 5 of $\mathcal{I}_{12}$ hold in $\mathcal{C}.s$.

  Since Item 3 of $\mathcal{I}_{12}$ holds in $\mathcal{C}$, to violate Item 3 of $\mathcal{I}_{12}$ in $\mathcal{C}.s$, R has to execute Line 2 with $R.g = G$ or execute Line 4 with $R.g \neq X.1$. However, since Items 1–2 of $\mathcal{I}_{12}$ hold in $\mathcal{C}$, these two cases are impossible. Therefore, Item 3 of $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

  To violate Item 4 of $\mathcal{I}_{12}$ in $\mathcal{C}.s$, R needs to execute Line 6 to Flip $Flag$. However, Item 3 implies that is impossible.

  Therefore, $\mathcal{I}_{12}$ holds in $\mathcal{C}.s$.

- If s is a step taken by the Writer W, then $PC_W = 13$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{13}$ holds in $\mathcal{C}.s$.

If $Flag = 1$, it is easy to notice that this case is the same as the case when W executes Line 11 with $Flag = 1$ and then goes to Line 13. Since we already proved $\mathcal{I}_{13}$ holds in that case, we can conclude that $\mathcal{I}_{13}$ holds in this case.

If $Flag = 0$, then Items 9–10 of $\mathcal{I}_{13}$ hold unconditionally. Since $\mathcal{I}_{12}$ holds in $\mathcal{C}$, we can easily figure out the rest of items of $\mathcal{I}_{13}$ hold in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.9 *If $PC_W = 13$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.9:

- If s is a step taken by a reader R, then $PC_W = 13$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{13}$ holds in $\mathcal{C}.s$.

  Similar to the proof of Claim 3.6, we can prove Items 1–4, 6, 7, and 11 of $\mathcal{I}_{13}$ hold in $\mathcal{C}.s$. Similar to the proof of Claim 3.7, we can prove Items 8–10 of $\mathcal{I}_{13}$ hold in $\mathcal{C}.s$.

  The only item left is Item 5. To violate it, R has to execute Line 5 when $W.c = 0$ and $Y = 0$, so that $Y \neq 0$ after this execution. However, since Item 5 holds in $\mathcal{C}$, we cannot have $W.c = 0$ and $Y = 0$ at the same time. Therefore, Item 5 and hence the whole $\mathcal{I}_{13}$ holds in $\mathcal{C}.s$.

- If s is a step taken by the Writer W with $W.c = 0$ in $\mathcal{C}$, then $PC_W = 7$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

  The only change that the execution of Line 13 causes is to make $G = W.s + 2$. Since we have $W.s = X.1$ in $\mathcal{C}$, $G = X.1 + 2$, i.e., Item 1 of $\mathcal{I}_7$ holds in $\mathcal{C}.s$. Since

Items 2, 4, 5, 6, and 7 of $\mathcal{I}_{13}$ hold in $\mathcal{C}.s$, we can figure out that Items 2-6 of $\mathcal{I}_7$ hold in $\mathcal{C}.s$. Therefore, $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W with $W.c \neq 0$ in $\mathcal{C}$, then $PC_W = 14$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{14}$ holds in $\mathcal{C}.s$.

  Like what we argued above, we can figure out Item 1 of $\mathcal{I}_{14}$ will hold in $\mathcal{C}.s$. Since Items 2, 4, 8, 9, and 11 of $\mathcal{I}_1$ and $W.c \neq 0$ hold in $\mathcal{C}$, it is obvious that Items 2–6 of $\mathcal{I}_{14}$ hold in $\mathcal{C}.s$. Therefore, $\mathcal{I}_{14}$ holds in $\mathcal{C}.s$.

Hence the proof. □

Claim 3.10 *If $PC_W = 14$ in configuration $\mathcal{C}$, then $\mathcal{I}$ holds in $\mathcal{C}.s$.*

Proof of Claim 3.10:

– If s is a step taken by a reader R, then $PC_W = 14$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_{14}$ holds in $\mathcal{C}.s$.

  Similar to the proof of Claim 3.6, we can prove Items 1-3 of $\mathcal{I}_{14}$ hold in $\mathcal{C}.s$. Similar to the proof of Claim 3.7, we can prove Items 4-6 of $\mathcal{I}_{14}$ hold in $\mathcal{C}.s$. Therefore, $\mathcal{I}_{14}$ holds in $\mathcal{C}.s$.

– If s is a step taken by the Writer W, then $PC_W = 7$ in $\mathcal{C}.s$. Therefore, we need to prove $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

  Since the execution of Line 14 only flips $Flag$ and $\mathcal{I}_{14}$ holds in $\mathcal{C}$, it is obvious that $\mathcal{I}_7$ holds in $\mathcal{C}.s$.

Hence the proof. □

3.2.2 Proof of the Properties

Based on the invariants above, we prove in this section that our single-writer multi-reader algorithms satisfies properties P1–P6 and achieves constant RMR complexity and constant shared space complexity.

It is obvious that this single-writer multi-reader algorithm only employs $O(1)$ shared space and satisfies Bounded Abort and Bounded Exit properties. In the following, we prove that this algorithm satisfies properties P1 and P3–P5, and that it achieves $O(1)$ RMR complexity in CC models.

Lemma 3.11 *This algorithm satisfies reader-writer exclusion.*

Proof of Lemma 3.11: $\mathcal{I}_{12}$ states that when the writer is in the CS (i.e., at Line 12), no reader is in the CS (i.e., at Line 3). Hence the proof. □

Lemma 3.12 *This algorithm satisfies starvation freedom.*

The proof of this lemma is based on the following claim.

Claim 3.13 *If a reader R is in the Try Section and has $G \geq 2 \vee G = R.g$ at some time t, then R is enabled after t, and $G \geq 2 \vee G = R.g$ holds until R has exited.*

Proof of Claim 3.13: Suppose we have $G \geq 2 \vee G = R.g$ at time t, for a reader R with $PC_R = 2$. If G remains unchanged, R will be able to get into the CS after executing Line 2. Note that G's value can only be changed at Line 8 and Line 13. According to $\mathcal{I}_{13}$, when W is about to execute Line 13, $W.s = X.1 \in \{0, 1\}$. Hence, $G \geq 2$ after Line 13 is executed. Thus, R can still get into the CS. According to $\mathcal{I}_9$, we have $|\{R|\ PC_R = 2 \wedge R.g \neq X.1\}| = 0$ and $G = X.1$ after W executes Line 8. Therefore, we know the reader R must have $R.g =$

$X.1 = G$. Thus, R still satisfies $G = R.g$ and hence can still get into the CS. Therefore, R is enabled after t. Moreover, $\mathcal{I}_9$ also states that $|\{R|\ PC_R \in \{3,4\} \wedge R.g \neq X.1\}| = 0$ and $|\{R|\ PC_R \in \{5,6\}\}| = 0$. The former predicate implies that, if R is at Lines 3–4, $R.g = X.1 = G$ after W executes Line 8. The latter one implies that W cannot execute Line 8 when R is at Lines 5–6. Therefore, $G \geq 2 \vee G = R.g$ always holds until R has exited. Hence the claim. □

Proof of Lemma 3.12: First we prove the writer W will not starve. Since W can wait at Line 7 and Line 11, we need to prove W will not wait forever at either of these two lines. We will analyze all possibilities below.

1.1) Suppose W waits at Line 7 and $Flag = 0$. Then according to $\mathcal{I}_7$, $|\{R|\ PC_R = 6\}| = 0$ as long as W is at Line 7. Since the only way for readers to change $Flag$ is to execute Line 6, we know $Flag$ will remain 0. Hence, W can pass Line 7 after W takes a step at any time. **1.2)** Suppose W waits at Line 7 and $Flag = 1$. **1.2.a)** If $Y = 0$, then, by $\mathcal{I}_7$, we have $|\{R|\ PC_R = 6\}| = 1$. Therefore, after the only reader at Line 6 takes a step, we have $Flag = 0$ and hence W can pass Line 7 by taking a step. **1.2.b**) If $Y \neq 0$, then, by $\mathcal{I}_7$, we have $Y = -|\{R|\ PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}|$, $|\{R|\ PC_R = 6\}| = 0$, and $G = X.1 + 2 \geq 2$. Since $X.1$ remains unchanged as long as W waits at Line 7, any reader $R \in \{R|\ PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}$ will finally go to Line 5 and increment Y. $\mathcal{I}_7$ also states $\forall R, PC_R = 5 \Longrightarrow R.g \neq X.1$. This implies that all readers R' with $R'.g = X.1$ must be at Line 2–4. Hence, these readers will find $R'.g = R'.g'$ after executing Line 4 and cannot get into Line 5 to change Y. On the other hand, any new reader R' will get $R'g = X.1$ after executing Line 1. Therefore, we can conclude that all the readers that can affect Y must be those that are already at Line 2–5 with $R.g \neq X.1$ before W waits at Line 7. Since

$Y = -|\{R|\ PC_R \in \{2,3,4,5\} \wedge R.g \neq X.1\}|$, Y will become 0 after all these readers execute Line 5. Now, we get back to case **1.2.a)** and know that W can pass Line 7.

2) Suppose W waits at Line 11. By similar arguments, we can prove the following facts based on the invariant $\mathcal{I}$.

- **2.1)** Once $Flag$ becomes 1, it will remain 1 before W goes through Line 11. Thus, W will not wait at Line 11 forever.
- **2.1.a)** If $Flag = 0$ and $Y = 0$, then there is only one reader at Line 6 and $Flag$ will become 1 after that reader takes a step.
- **2.1.b)** If $Flag = 0$ and $Y \neq 0$, then after all the current readers R's with $R.g \neq X.1$ finally execute Line 5, we have $Y = 0$ and know that $Flag$ will become 1.

According to these facts, we can conclude that W will not wait at Line 11 forever. Hence, W won't starve.

Since W won't starve, W can finally execute Line 13 and make $G \geq 2$. Thus, any reader R waiting at Line 2 will then find $G \geq 2 \vee G = R.g$. By Claim 3.13, R is now enabled and hence won't starve.

Lemma 3.14 *This algorithm satisfies FIFE among readers.*

Proof of Lemma 3.14: The doorway for the readers is line 1. Assume this algorithm does not satisfy FIFE. That is, there exists a scenario where a reader R executes Line 1 before another reader R' does, while R' is enabled earlier than R. Since R comes in the Try Section before R', at the moment R' becomes enabled, R and R' must be both in the Try Section, i.e., $PC_R = PC'_R = 2$.

If $R.g = R'.g$, then at the moment R' is enabled, we know $R'.g = R.g = G$. By Claim 3.13, this implies that R is also enabled at the same time, contradicting the assumption that R' is enabled earlier than R.

Suppose $R.g \neq R'.g$. Since R and R' get the values of $R.g$ and $R'g$ respectively from $X.1$ at Line 1, and $X.1$ can only be changed at Line 9, we can conclude that the writer W executes line 9 to flip $X.1$ at a time t between the time R executes line 1 and the time R' executes line 1. Since $X.1$ is a two-valued variable and $G = X.1$ holds when W is about to execute Line 9, this implies that $G = R.g$ holds either before t or after t. By Claim 3.13, R is enabled once $G \geq 2 \vee G = R.g$. Therefore, R is already enabled even before R' enters the Try Section, a contradiction.

Hence, this algorithm satisfies FIFE. □

Lemma 3.15 *This algorithm satisfies concurrent entering.*

Proof of Lemma 3.15: If the writer W is in the Remainder Section, i.e., $PC_W = 7$, then, by $\mathcal{I}_7$, $G = X.1 + 2 \geq 2$. Therefore, for all readers R at line 2, $G \geq 2 \vee G = R.g$. By Claim 3.13, This implies that all readers in the Try Section are enabled. Therefore, if W stays in the Remainder Section, every reader can enter the CS in a bounded number of its own steps. Hence, this algorithm satisfies concurrent entering. □

Lemma 3.16 *This algorithm has $O(1)$ RMR complexity in the CC model.*

Proof of Lemma 3.16: As we argued in the proof of Lemma 3.12, if W spins at Line 7 or Line 11, then $Flag$ can be changed at most once by readers, and after that change $Flag$ will remain a value such that W can pass Line 7 or Line 11. Combining this and the fact that all other lines for the writer are executed only once in a write attempt, we can conclude that the writer has $O(1)$ RMR complexity in the CC model.

Suppose a reader R spins at line 2. Since the writer cannot starve, it will execute Line 13 to make $G \geq 2$. As Claim 3.13 showed, R is now enabled and can get into the CS by taking one step. Moreover, G can only be undated by the writer. Therefore, it only takes $O(1)$ RMRs for R to spin at Line 2. Combining this and the fact that all other lines for readers are executed only once in a read attempt, we can conclude that the readers have $O(1)$ RMR complexity.

Hence, this algorithm has $O(1)$ RMR complexity in the CC model. □

Theorem 3.17 (Abortable Single-Writer Multi-Reader Starvation-Free Algorithm) *The algorithm in Figure 1 satisfies properties P1–P6. The RMR complexity of this algorithm in CC model is* $O(1)$. *This algorithm employs* $O(1)$ *number of shared variables that support read, write, and fetch&add operations.*

4 Transformation from Single-Writer Algorithm to Multi-Writer Algorithm

We convert our single-writer algorithm into a multi-writer algorithm using a simple transformation similar to the one suggested by Bhatt and Jayanti [3](see Figure 3). Here readers simply execute the Reader procedure of the underlying abortable single-writer algorithm. Writers, on the other hand, first obtain an abortable mutex lock M and only then execute the Writer procedure of the underlying abortable single-writer algorithm. When a writer exits or aborts from the underlying Writer procedure, it releases the lock immediately after the last step of the Exit Section or Abort Section, respectively.

If M is an abortable mutex lock satisfying starvation-freedom, like the one in [11],

M: abortable Mutex Lock

Reader	**Writer**
SW-Reader()	acquire(M)
	SW-Writer()
	release(M)

Figure 3: Transforming a single-writer multi-reader algorithm SW to a multi-writer multi-reader algorithm.

we claim that the transformation in Figure 3 gives an abortable multi-writer multi-reader algorithm that satisfies P1–P6. More generally, given any abortable single-writer multi-writer algorithm, we can use this transformation to construct an abortable multi-writer multi-reader algorithm. This is trivially true because of two facts: (1) since the lock M ensures that only one writer accesses the underlying writer procedure at any time, the underlying single-writer multi-reader algorithm works as if it is in a single-writer multi-reader system; and (2) when a writer leaves the underlying writer procedure, it always releases the lock M so that the next writer can acquire M and then enter the underlying writer procedure, and hence no writer can starve if it doesn't abort. A formal proof can be found in [3].

What's more, if the lock M satisfies FCFS property, then it is obvious that the abortable multi-writer multi-reader algorithm satisfies FCFS among writers. Since our single-writer multi-reader algorithm only has $O(1)$ RMR complexity and employs $O(1)$ shared space, the RMR and shared space complexities of the multi-writer multi-reader algorithm depend on those of the abortable mutex lock M.

Theorem 4.1 (Abortable Multi-Writer Multi-Reader Starvation-Free Algorithm) *The algorithm in Figure 3 satisfies properties P1–P6. The RMR and shared space complexities of*

this algorithm are $O(r(M))$ and $O(s(M))$, where $r(M)$ and $s(M)$ are the RMR and shared space complexities of the abortable starvation-free mutex lock M used in this algorithm, respectively. If the M lock satisfies FCFS, then this algorithm satisfies FCFS among writers (property P7).

Therefore, if the lock M used in Figure 3 is one of the $O(\log n)$-RMR locks in [11] and [14], we construct an abortable, starvation-free reader-writer lock of $O(\log n)$ RMR complexity. To our knowledge, this is the first abortable, starvation-free reader-writer lock.

References

[1] James H. Anderson, Yong-Jik Kim, and Ted Herman. Shared-memory mutual exclusion: major research trends since 1986. *Distrib. Comput.*, 16(2-3):75–110, 2003.

[2] T. E. Anderson. The performance of spin lock alternatives for shared-memory multiprocessors. *IEEE Trans. Parallel Distrib. Syst.*, 1(1):6–16, 1990.

[3] Vibhor Bhatt and Prasad Jayanti. Constant rmr solutions to reader writer synchronization. 2010. Submitted to *PODC '10 : Proceedings of the Twenty-Ninth annual symposium on Principles of distributed computing.*

[4] Vibhor Bhatt and Prasad Jayanti. Specification and constant rmr algorithm for phase-fair reader-writer lock. In *ICDCN '11: Proceedings of the 12th international conference on Distributed computing and networking*, pages 119–130, 2011.

[5] Bjorn B. Brandenburg and James H. Anderson. Reader-writer synchronization for shared-memory multiprocessor real-time systems. In *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*, pages 184–193, Washington, DC, USA, 2009. IEEE Computer Society.

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

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

[8] Michael J. Fischer, Nancy A. Lynch, James E. Burns, and Allan Borodin. Resource allocation with immunity to limited process failure. In *SFCS '79: Proceedings of*

the 20th Annual Symposium on Foundations of Computer Science, pages 234–254, Washington, DC, USA, 1979. IEEE Computer Society.

[9] Vassos Hadzilacos. A note on group mutual exclusion. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 100–106, New York, NY, USA, 2001. ACM.

[10] Vassos Hadzilacos and Robert Danek. Local-spin group mutual exclusion algorithms. volume 3274, pages 71–85. Springer Berlin / Heidelberg, 2004.

[11] Prasad Jayanti. Adaptive and efficient abortable mutual exclusion. In *PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 295–304, New York, NY, USA, 2003. ACM.

[12] Yuh-Jzer Joung. Asynchronous group mutual exclusion (extended abstract). In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 51–60, New York, NY, USA, 1998. ACM.

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

[14] H Lee. Fast local-spin abortable mutual exclusion with bounded space. In *OPODIS '10: Proceedings of the 14th conference on Principles of distributed systems*, pages 364–379. Springer-Verlag, 2010.

[15] Yossi Lev, Victor Luchangco, and Marek Olszewski. Scalable reader-writer locks. In *SPAA '09: Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 101–110, New York, NY, USA, 2009. ACM.

[16] Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7:321–359, November 1989.

[17] John M. Mellor-Crummey and Michael L. Scott. Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Trans. Comput. Syst.*, 9(1):21–65, 1991.

[18] John M. Mellor-Crummey and Michael L. Scott. Scalable reader-writer synchronization for shared-memory multiprocessors. In *PPOPP '91: Proceedings of the third ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 106–113, New York, NY, USA, 1991. ACM.

[19] M. L. Scott. Non-blocking timeout in scalable queue-based spin locks. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing*, pages 31–40, 2002.

[20] M. L. Scott and W. N. Scherer III. Scalable queue-based spin locks with timeout. In *Proceedings of the 8th Symposium on Principles and Practice of Parallel Programming*, pages 44–52, 2001.

[21] C. K. Tang. Cache system design in the tightly coupled multiprocessor system. In *Proceedings of the June 7-10, 1976, national computer conference and exposition*, pages 749–753, New York, NY, USA, 1976. ACM.

[22] Nan Zheng. Constant-rmr abortable reader-priority reader-writer algorithm. Technical Report TR2011-685, Dartmouth College, Computer Science, Hanover, NH, June 2011.