THE INSPECTION OF CONCURRENT PROGRAMS

USE OF TABULAR EXPRESSIONS

IN

THE INSPECTION OF CONCURRENT PROGRAMS

By

XIAO-HUI JIN

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Master of Science

McMaster University

MASTER OF SCIENCE (2004) McMaster University

(Computing and Software) Hamilton, Ontario

TITLE: Use of Tabular Expressions in the Inspection of Concurrent Programs

AUTHOR: Xiao-Hui Jin

M. Eng. (Beijing University of Aeronautics and Astronautics)
B. Eng. (ShenYang University of Aeronautical Engineering)

SUPERVISOR: David L. Parnas

NUMBER OF PAGES: x, 234

Abstract

This thesis presents a systematic, rigorous inspection approach for concurrent programs. The approach has been successfully applied to a classic concurrent program of the Readers/Writers problem.

In the inspection process, we rewrite the concurrent program by assigning each primitive statement a label; the transfer of control from statement to statement is made explicit. Auxiliary variables are used to record extra information for inspection without affecting the original intent of the program. The resulting program is a non-deterministic sequential program with the same behavioral effect as the original concurrent program. The rewritten program is then examined through checking the truth-value of the system invariant that fully captures program structure. A decreasing quantity of the program states is also used to show the clean completion of the program.

We use tabular expressions, program-function tables, to describe the function of the program. Each column in the table is inspected individually; the program is 'divided' into small components to be 'conquered' with ease. The correctness of the whole program is implied (evaluated) by the correctness of the columns examined through the inspection.

Acknowledgements

First of all, I thank God for His mercy and grace upon me.

I give great thanks to my supervisor, Dr. David L. Parnas, for his wisdom, guidance, and help throughout the course of this research project. I have learned a lot from him, not only in software engineering but also in daily life.

I am very grateful to Dr. Sanzheng Qiao and Dr. Mark Lawford for their reviews and useful comments on my thesis. Dr. Alan Wassyng has been a source of valuable advice during the final stage of this thesis. My thanks also go to Dr. Farmer, Dr. Kahl, Dr. Baber for their kind suggestions on my advancing the thesis.

Dr. Ying Jin from Limerick deserves special thanks for her intelligent and generous help with this thesis. Continuous encouragement and friendship from Ireland, especially from Bing Yan, Victor Liu, Lisa Lu, and Lillian Parnas are great inspiration for me to persevere.

Thanks to Doris Burns, Laurie LeBlanc, and Ita Page for their kind help and encouragement, especially for making my research in the University of Limerick possible. Thanks to all the people in the CAS Department Office for their nice help.

Thanks to all of the graduate students in Room 223, SQRL Lab, for their support and their sharing of thoughts during these Mac years.

I thank the Campus Christian Fellowship for their support and sincere prayers. The Hamilton Mandarin Alliance Church has been a source of spiritual-help to keep my heart in tune. Very special thanks to Dr. Tony Quan, who often encouraged me in good spirit especially during the later hard time; his kindness and patience in proofreading and revising this thesis is greatly appreciated.

I would like to thank Wei Qiao for his care, understanding, and support, which help me change my life in amazing ways.

Finally, I cannot thank enough to my mom, my dad, and my two dear sisters for their deep and unfailing love. This thesis is dedicated to them.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In spite of many years' effort to improve software quality, much work still needs to be done to develop more effective software inspection approaches.

People always experience more difficulties when inspecting concurrent software. This is because many things are happening at the same time in concurrent systems, making it easy to overlook some of the possible event sequences. Consequently, in spite of detailed review and analysis, the programs are often wrong. Many verification techniques can be deployed to check program's correctness, however, they often make simplifying assumptions that are not valid for real programs. For example, although model checking can be used to verify a concurrent system, the checking algorithm usually only deals with fixed point cases, leaving many practical problems out of the application scope. Therefore, there is a need for a sound and systematic inspection procedure, supported with appropriate computer-aided tools, which is able to account for all cases and possible event sequences during inspection and review.

This thesis proposes a systematic, rigorous approach to inspecting concurrent software. It is built on a foundation of mathematical relations that describe program

behavior [44][50][52]. The use of tabular expression in describing those relations makes inspection easier and more systematic [44][48][50]. Such tables provide strong support for reviewing and inspecting a system to make sure that it performs as intended [42][48][53]. We use a classic example of concurrent programming, the Readers/Writers problem [15], to demonstrate the procedure in detail.

1.2 Our Approach

The key idea in our approach is the policy of "divide and conquer" [48][53], which enables the inspector to look at many simple cases without overlooking anything. The correctness of the whole program is implied by the correctness of the examined program components. Tabular notations are used to describe the function of the program. First, the table is confirmed to be complete and consistent; subsequently, it can be considered column-by-column and row-by-row. A break can be taken at the end of any entry's consideration without danger of overlooking any cases.

In our approach, we rewrite each sequential program by giving each primitive (uninterruptible) statement in the program a label. Transfer of control to each statement is made explicit by "go to" statements; these are treated as assignments to a label variable. The execution of each statement is conditioned depending on the value of the label.

In theory, any conditional statement in the rewritten program can be executed at any time. However, the execution will cause a change in the data state only if the statement's control condition is met, i.e. the label variable's value is appropriate. When several statements in different processes could be executed, we treat the choice as random, with the exception that no statements will be executed unless there has been some change in the data state since its most recent execution. This prevents an endless execution of statements without affecting the data. It is necessary to prove that there will always be a statement to be executed. The most creative and difficult step in the inspection is the identification of a sufficiently strong invariant for the system of programs. The inspection procedure verifies that the invariant is maintained during the execution and that the program will not stall.

Each program statement, when executed, may change the state of the data structure, and can be described by a mathematical relation that maps from a starting state to a final state of the program [49]. For a sequential program, given the initial state, the program describes one or more sequences of state changes. Thus, the program behavior can be described by a set of ordered pairs (starting state, ending state) that defines a relation. When computing the relations that describe large programs, we will use a variety of operations such as sequential composition and conditional composition [54]. However, a concurrent program begins execution with an initial state and advances with an interleaving of a set of sequential statements, which are executed by the constituent processes [6][13]. By including the control variable (statement label) in the state, we can describe the effect of the program behavior using the union operation for the relations rather than the composition. It only applies to the program as rewritten to transform each statement into a conditional statement that will have no effect when executed except when it is "its turn".

The techniques used in this paper also include the concept of program invariants [24][27]. These are predicate expressions that are true after each complete execution of the program statements, provided that they are true before the execution. This concept was originally developed for the verification of loops; we extend its use here to analyze concurrent and non-deterministic programs.

The requirements of programs can be described by predicate expressions, including a set of invariant predicates of the program, which must be true throughout the execution of the program. Using a tabular representation of the program, the program-function table [44], we will show that the proposed approach with the aforementioned techniques can be effective and reliable in program inspection.

1.3 Organization of the Thesis

Chapter 2 contains a brief introduction of the Tabular Expression used in our approach. Program-function tables are introduced and explained using a basic example.

Chapter 3 provides an overview of concurrent programs, and explains the concept of executing concurrent programs. Synchronization mechanisms are introduced with emphasis on the semaphore concept. The von Neuuman's Principle (program state and data state are interchangeable) is briefly explained with a special focus on the treatment of concurrency.

Chapter 4 describes the basic inspection concepts of both sequential and concurrent programs. The use of tabular notation based on the principle of "divide and conquer" and the program-function table are explained in detail.

Chapters 5 - 8 demonstrate our approach and the applicability of the technique by applying it to a classic example in concurrent programming. Techniques for performing the inspection given the requirement/implementation of the program are presented.

Chapter 9 presents the conclusions from this research project and discusses possible future work.

Chapter 2

Tabular Expressions

The importance of precisely documenting software products is widely recognized and understood in software engineering. Mathematical notation, allowing for precise and rigorous documentation, is highly recommended to describe the requirements and behaviors of software systems [49]. However, using conventional mathematical notation to document real software products often results in function descriptions that are complex, lengthy, and hard to understand [44]. Consequently, the mathematical notation is commonly used in academic papers but is very rarely used in industry due to the associated complexity.

In 1977, based on practical experience, Parnas et al. [44][47][50][52] proposed the use of multi-dimensional mathematical notations, called tables, to represent mathematical functions and relations in documenting software systems. This method has proven very useful for precise and concise documentation and inspection of software systems.

This technique was first developed in the documentation for the United States NRL A-7E aircraft project [59] in the late 1970s. The shutdown system of the Darlington Nuclear Power Generation Station in Ontario, Canada is another example of this technique's application. In this work, tables were used to conduct thorough inspections of safety-critical programs, in which numerous discrepancies were found and required correction [48].

Tabular expressions were initially used on an ad hoc basis; new types of tables were introduced whenever they appeared to be useful. Parnas [44] provided the first formalization of tabular expression, describing ten classes of tables by giving their syntax and semantics. Predicates and functions were used as the definitions of each proposed type of table. Later, a model for the semantics of tables was presented by Janicki and Khedri [28][29]. A more generalized model of tabular expressions was presented in Abraham's thesis [2]. These models covered all of the table classes from Parnas [44] and some new classes that had not been considered before.

2.1 Why Tabular expression

Tabular expression consists of multi-dimensional expressions, called tables; they are equivalent in expressive power to expressions written in the traditional manner. Although there is no increase in theoretical expressive power over that of conventional expression, tables have many practical advantages [29][44][47]: they enable the detailed design information to be organized in a way that is easier to use than unstructured traditional representations.

Tables are particularly well suited for describing the kinds of conditional relations that frequently occur in program descriptions and specifications. The structure of tables is particularly advantageous when many cases have to be considered in a problem, and makes it unnecessary to continually repeat common sub-expressions. Moreover, to developers, testers, and reviewers (or inspectors), the table format provides strong support by making all of the possible cases explicit, which allows checks for consistency and completeness. In practice, a variety of situations often need to be considered in order to make sure that the program behaves properly. Therefore, it is very likely that inspectors overlook the same cases that the designers failed to consider. Tables with a particular column-row structure can help to overcome these problems.

Most importantly, tables can facilitate an analysis for completeness and consistency in the inspection procedure. Using a tabular structure, when the set of columns of

the table is complete with no overlapping, makes the process of inspection much easier.

2.2 Basic Concepts

Tables are multi-dimensional expressions constructed from simpler components, including conventional expressions and grids [44]. The main grids and header grids are indexed by sets of cells that contain predicate expressions. Cells in grids can consist of tabular expressions, when useful.

For example, a typical table shown in figure 2-1 (taken from [30][44]) consists of a main grid G and header grids H1, H2, which are subdivided by cells containing predicate expressions:

		$y = 10$	$y > 10$	$y < 10$	H_1
H_2	$x \geq 0$	0	y^2	$-y^2$	
	$x < 0$	x	$x + y$	$x - y$	G

Figure 2-1 A Typical Table

Each table is equivalent to a conventional mathematical expression [44], and thus can be treated as an expression. The above table describes an expression in the liberal mathematical notation shown below:

$$f(x,y) = \begin{cases} 0 & \text{if } x \geq 0 \wedge y = 10 \\ x & \text{if } x < 0 \wedge y = 10 \\ y^2 & \text{if } x \geq 0 \wedge y > 10 \\ -y^2 & \text{if } x \geq 0 \wedge y < 10 \\ x + y & \text{if } x < 0 \wedge y > 10 \\ x - y & \text{if } x < 0 \wedge y < 10 \end{cases}$$

Using classic predicate logic, the equivalent expression is given as follows:

$$(\forall x, (\forall y, ((x \geq 0 \wedge y = 10) \Rightarrow f(x,y) = 0) \wedge ((x < 0 \wedge y = 10) \Rightarrow f(x,y) = x) \wedge$$
$$((x \geq 0 \wedge y > 10) \Rightarrow f(x,y) = y^2) \wedge ((x \geq 0 \wedge y < 10) \Rightarrow f(x,y) = -y^2) \wedge$$
$$((x < 0 \wedge y > 10) \Rightarrow f(x,y) = x + y) \wedge ((x < 0 \wedge y < 10) \Rightarrow f(x,y) = x - y)))$$

Figure 2-2 The Equivalent Expression for Table in Figure 2-1

Detailed and formal discussions of tabular expression and its definitions can be found elsewhere [28][30][39][44]. This thesis explains and uses only one of the table types, the program-function table.

2.3 The Program-Function Table

A program-function table is essentially a mixed-vector table [44]. To explain this concept, we consider a simple example problem: describing a program that finds the maximum of two positive values stored in program variables, and assigns the value *false* to a flag when they are equal. Figure 2-3 is a description of the program using a program-function table.

	'x >'y	'x ='y	'x < 'y
strictmax' |	strictmax' = 'x	true	strictmax' = 'y
find' =	*true*	*false*	*true*

$\wedge NC^1(x, y)$

Figure 2-3 An Example of Program-Function Table

[1] The NC symbol stands for Not Changed ([44]), and is used to express the fact that some variables do not change their values during the execution of a program: $NC(V_1, ..., V_k) \Leftrightarrow (V_1' = 'V_1) \wedge ... \wedge (V_k' = 'V_k)$

Before **and** ***After*** **Values**

The ***before*** and ***after*** symbols in the table are used according to the notational conventions from Parnas [42][49]:

Let **P** be a program, and let ($\mathbf{V_1}$, ..., $\mathbf{V_k}$) be the variables in P that constitute its data structure. Then

> $\mathbf{V_i}$' denotes the value of the program variable $\mathbf{V_i}$ after an execution of **P**;
> '$\mathbf{V_i}$ denotes the value of the program variable $\mathbf{V_i}$ before an execution of **P**.

The program-function table is a representation of the characteristic predicate of the program function. The header at the top of the table (see figure 2-3) contains predicates (conditions) that partition the domain of the function and distinguish the three situations. The header at the left of the table contains the variable names of a program. Each row in the main grid of the table corresponds to one program variable, describing the ***after*** value that this variable must have upon termination when the variable's ***before*** value satisfies the column's heading predicate.

The row headings of the program-function table not only identify the variable whose value is described in each row, but also indicate how that variable will be described. The symbol "=" in the row header indicates that the grid elements in that row must be expressions by which the values of that variable will be evaluated. When the symbol "|" appears instead of "=", the variable's value must satisfy a predicate given in the appropriate cell in the main grid [52].

A program-function table represents a predicate, which can be a term of a conjunction with another table or any Boolean expression. The equivalent Boolean expression of figure 2-3 is shown below:

$$(((\text{`x} > \text{`y}) \wedge (\text{max'} = \text{`x}) \wedge (\text{find'} = \textit{true})) \vee ((\text{`x} = \text{`y}) \wedge (\text{find'} = \textit{false})) \vee$$
$$((\text{`x} < \text{`y}) \wedge (\text{max'} = \text{`y}) \wedge (\text{find'} = \textit{true}))) \wedge (\text{x'} = \text{`x} \wedge \text{y'} = \text{`y})$$

Figure 2-4 The Equivalent Boolean Expression for Table in Figure 2-3

The table represents a set of simpler Boolean expressions rather than a single complex expression, which is much easier to read than the conventional expressions. The table eliminates nested pairs of parentheses and reveals the intended structure of the Boolean expression. In addition, in order to find out what must happen in a specific case, one can select the row and column of interest from the table without needing to understand the whole expression.

For large tables involving many cases, many variables, and long identifiers, the advantages of the tabular expression is more pronounced. For instance, when long variable names are used in the definition of the conditions, visual clutter can be reduced to a minimum by using dotted lines in the header of the table to indicate common sub-expressions.

The program-function table in figure 2-3 can be rewritten using dotted line technique in the following equivalent form:

	'x			
	>'y	='y	< 'y	
strictmax' |	strictmax' = 'x	***true***	strictmax' = 'y	
find' =	***true***	***false***	***true***	∧ NC (x, y)

Figure 2-5 A Program-Function Table Using Dotted Lines

2.4 Completeness and Disjointness Properties

Tables should satisfy the properties of completeness and disjointness [43]. A program-function table should be built and used with this consideration in mind.

Disjointness of a Program-function Table:

This property requires the conjunction of pair-wise column header expressions be ***false*** for the whole header of the table, i.e. the columns of the tabular presentation of the program function do not overlap. It is defined to make sure that each column deals with a disjoint subset of the domain of the program function.

Completeness of a Program-function Table:

This property requires the disjunction of all of the column header expressions be ***true***, i.e. the union of all header expressions of the table is always ***true***. It is defined to confirm that the table covers the entire domain of the program function. If the domain of the function described is intended to be partial, ***true*** should be replaced by the characteristic predicate of the intended domain.

The program-function table shown in figure 2-3 has the disjointness property if the following is ***false***:

$$(((`x >`y) \wedge (`x=`y)) \vee ((`x >`y) \wedge (`x <`y)) \vee ((\ `x=`y) \wedge (`x <`y)))$$

The program-function table shown in figure 2-3 is complete if the following is ***true***:

$$((`x >`\ y) \vee (\ `x=`y) \vee (`x <`y))$$

These two properties of the program-function table ensure that in each state of the program in execution, there must be one column condition in the tabular representation that is true, and that there is no state in which two column conditions are true at the same time.

Chapter 3

Overview of Concurrent Programs

This chapter presents a brief overview of concurrent programs that is not aimed at covering all of the literature on the subject, but only those works related to understanding the techniques proposed in this thesis.

3.1 Execution of Concurrent Programs

3.1.1 Process

A concurrent program specifies two or more sequential programs that may be executed concurrently as parallel processes. A sequential program specifies execution of a list of statements in sequence; its execution is called a process [5][6][13][14].

In a concurrent program, a set of processes can be executed by either making them share one or more processors, or running each on its own processor; in the limiting case of a single processor, all processes are interleaved or time-shared on the same processor. However, one can understand concurrent programs by analyzing their

components' sequential processes and their interactions without knowing how the processes are executed.

This is possible because the effect of a sequential program is independent of the speed of execution [26]. No assumption is made of the speed of the concurrently running processes so that the correctness of the program is independent of their physical execution environment.

3.1.2 Process Execution Interleaving

When a sequential program is executed, there is a single thread of control: the program counter starts with the first atomic action and moves through the atomic actions that follow. However, when a concurrent program is executed, there are multiple threads of control, each of them assigned to a particular component process. Each component process is sequential and its execution often has to be interleaved.

The component processes of a concurrent program may exhibit extremely complicated behavior because of their interaction. This makes the associated analysis more difficult. Moreover, we must consider the cooperation of the component processes so as to ensure proper communication and synchronization. (Please refer to Section 3.2 for more discussion).

A sequential program delivers the same result each time it is executed for a given set of input data, whereas a concurrent program may take different execution paths when executed repeatedly even for the same input data. This leads to the non-deterministic property of concurrent programs. The execution interleaving in a concurrent program may have many possibilities, making review and inspection infeasible in practice. If contemplating the individual execution interleaving is impracticable, one must seek methods that allow for a thorough analysis of all possible executions. In fact, program text is a succinct description of the entire set of executions of the program. Therefore, direct analysis on the program text, rather

than on the executions that the program encodes, has the potential to circumvent many problems that limit the effectiveness of the inspection.

3.1.3 Equivalency Characteristic of the Execution

A concurrent program begins execution with an initial state, and advances with interleaving of a set of sequential statements executed by its constituent processes [6][13]. Each process in the program executes at an unknown speed and transforms the state by executing a sequence of primitive statements. Primitive statements are those that can be executed atomically. Once a process starts executing a primitive statement, another process cannot influence its execution or observe any intermediate point of the execution.

The execution of a primitive statement is an indivisible state change; one cannot observe a partial change of state. The concurrent execution of a set of sequential statements actually proceeds with some arbitrary, serial order. For example, if two primitive statements – we shall call them statement Sx_i, which is the i^{th} statement of process x, and Sy_j, the j^{th} statement of process y – are executed concurrently, the net effect is that either Sx_i follows Sy_j, or Sy_j follows Sx_i. Therefore, the execution of a concurrent program is equivalent to executing a single, non-deterministic program. The non-deterministic aspect arises when there is no certain order of execution of the processes, because one does not know which process will be executed next if several processes are ready to be executed.

3.1.4 Program Variables

The execution state of a program at any point consists of values of program variables, which consist of two types: one is defined explicitly by the programmer, while the other contains hidden state information (also called auxiliary variables [25], e.g. the program counter for each process). Variables are shared when defined

globally and can potentially be accessed by multiple processes. Usually they are used for process interaction and manipulated in a critical section.

In order to trace the execution of the processes, we can use auxiliary variables in the inspection of concurrent programs. Hailpern [25] introduced auxiliary variables to record the program information needed for a correctness proof, which is not available from the program's code. In this thesis we use auxiliary variables to describe the program counter for each process.

3.2 Communication and Synchronization

Processes in a concurrent program, when executed, cooperate by communicating and synchronizing.

Communication allows one process to influence the execution of another by exchanging data with one another or competing for exclusive access to the shared resources. There are two methods of communication: either a process writes the state of a shared variable that is to be read by another process, or a process sends a message that is to be received by another process.

When processes interact, not all interleavings are acceptable. This gives rise to the need for synchronization. Synchronization plays a fundamental role in preventing undesirable interleaving among the component processes of concurrent programs. Mutual exclusion is an important form of synchronization that occurs in concurrent programs [5][6][13][14]. It is widely used to ensure that, at most, one process at a time is being executed in the critical section, i.e. actions are grouped into the critical section with no interleaving during execution.

3.2.1 Synchronization Mechanisms

Synchronization mechanisms are employed to correctly control the interaction of processes. There are many kinds of mechanisms proposed in the literature, including the semaphore, the most important one. Many of the semaphore's proposed variations were developed in the following decades after the concept had been introduced. Another popular synchronization mechanism is "monitor," which is also applied in many modern concurrent systems[5][6].

In this thesis, we use one of the solutions of a classic synchronization problem, the Readers/Writers problem [15], as an example to demonstrate the proposed approach to inspection of such systems. Since the problem was solved using the semaphore primitives, we give a brief description of the semaphore mechanism in the next section. More details can be found in [16][18][19][32].

In the literature, researchers often use classic synchronization problems to compare and contrast synchronization mechanisms. The Readers/Writers problem, proposed by Courtois, Heymans, and Parnas, is one such example [15]. We chose this problem for demonstration because it is an eminently practical problem. Chapter 5 presents more detail and analysis of this problem.

3.2.2 Semaphore

The semaphore was invented by Dijkstra in 1968 [19] as a tool for implementing mutual exclusion in the development of "THE" system.

The basic idea of a semaphore is to provide limited access to shared resources by placing a guard (called a semaphore) around the codes accessing the resources [18]. A semaphore is a special-purpose integer variable allocated in a scope that is accessible to the parallel processes in a concurrent program; it is initialized with a

value of "0" or "1" prior to the commencement of the processes [19]. The semaphore variable is usually implemented as a counter with a waiting list, and is accessed by the parallel processes only via two specific operations: P-operation and V-operation.

Definition of semaphore[2]:

A semaphore **sem** consists of two function parts in use:

(1) an integer counter, **sem.cnt**, which is the value of the counting semaphore;
(2) a set, **sem.set**, which is the set of waiting processes blocked by P-operation.

If **sem.set** is not empty, then **sem.cnt** is negative and the magnitude of **sem.cnt** is the cardinality of **sem.set**, which is the number of processes waiting on semaphore **sem**, i.e., a negative value of **sem.cnt** indicates that we have |**sem.cnt**| waiting processes to wait for an invoking operation from another process. If **sem.cnt** is zero, then the **sem.set** is empty.

A semaphore **sem** may be accessed by parallel sequential processes, each of which is assigned an identifier to identify the process in use. The elements of **sem.set** are identifiers of the processes blocked by the P-operation and waiting for being invoked by a V-operation.

The operations of semaphore:

P-operation, denoted as **P(sem)**, decreases the semaphore value by 1: if the resulting value of semaphore **sem** is non-negative, the performing process can advance with the execution of its next statement; otherwise, the process is suspended and booked on the waiting list associated with the semaphore **sem**.

[2] Please note that this is the usual implementation of a semaphore but variations are possible.

V-operation, denoted as **V(sem)**, increases the semaphore value by 1: if the resulting value of semaphore **sem** is positive, the V-operation in question has no further effect; otherwise, one of the waiting processes is invoked and is ready to continue with its execution.

The P-operation includes a potential delay of the performing process, whereas the V-operation may activate one of the waiting processes and represents the removal of a barrier. The indivisibility of P/V-operations ensures the integrity of the values of semaphores [19]. In fact, semaphore operations embody two distinct concepts: one is suspension and wake-up of processes; the other is counting. These two tasks are related when the counter reaches a certain value and the execution of one process results in the state change of another process [11][19].

3.3 Von Neumann Principle

The von Neumann Architecture was created when John von Neumann and several other computer pioneers first spelled out the requirements for a general-purpose electronic computer [60]. In their project plan, they intended to build a new kind of computer using the "stored program" concept. Their success resulted in the von Neumann Architecture's wide use in a class of computers, including many of those existing today.

The key point of von Neumann Architecture is that, instead of wiring in program instructions as part of the machine, the "stored program" computer keeps its specific instructions (programs) in its memory, storing the information in the same manner as it would store any other information (data). In other words, the program and data are both stored in the same memory, instead of storing them in separate memories.

This principle was later expanded to allow control information (logic) to move from program into data, or vice versa; that is, the program state and data state are interchangeable. Harlan Mills illustrated this concept by showing how to describe a

Turing machine in two different views [38][39]. He demonstrated that the control state (program) is equivalent to the data state.

Although concurrent programs do not behave in the same way as sequential programs, any such concurrent program is composed of a set of sequential processes in execution. Therefore, a concurrent program can be analyzed by applying the same concept described above. In this thesis, we extend the von Neumann Principle by rewriting the Readers/Writers programs and moving the program state information into the data state.

Chapter 4
Inspection Using Tabular Expressions

4.1 Overview of Inspection

4.1.1 Why Inspection?

Almost all of the software in use today contains serious errors. Software quality assurance is therefore one of the biggest concerns for all major software developers. The main task is to get rid of bugs before products are released, and to remove errors from products already in use.

Researchers have responded to these problems by studying methods of formal correctness verification for programs [53]. The verification method focuses on how to prove program correctness with the same degree of rigor that we apply to mathematical theorems. The work often involves long, complex expressions and thus is tedious and error-prone. Because of the exhaustive mathematical treatments, the process is very time consuming and much patience and effort are expected from the participants. Therefore, formal verification is not practical and actually rarely done [53].

Software inspection is a type of quality review process, which was first described in the 1970s by Fagan when at IBM. This has become an essential technique for

improving software quality and productivity. Since its introduction, several variants of the technique have been proposed to further improve inspection performance[3]. Basically, a common inspection procedure includes three steps: preparation, collection, and repair [7]. First, each reviewer reads the artifact separately and detects as many defects as possible. Next, the defects are collected, usually at a team meeting. They are then sent to the artifact's author for repair. The entire process may be repeated one or more times when necessary.

One of the technique's main benefits is that it can be applied to any artifacts produced during the development stage, for instance, code, design, and program requirements [3]. This feature is important for detecting defects in time and removing systemic bugs before the product is released. Comparatively, inspection has certain features that make it more practical in reality than formal verification; the three main causes are that [53]

- The inspector can inspect the code itself, not just abstract models of it;
- Inspection does not require a training investment, which is must for verification;
- Inspection does not require the time and the formula manipulation ability that is needed in verification of typical programs.

Inspection can not only lead to certification of the quality of a program by detecting errors in its code, but also help ensure that a correct system is built and well maintained [53]. People do not need to wait until code is finished to reap inspection's benefits. Early inspection of system requirement documents and other system documents, which are generated later during the design process, can help insure the quality of the system. In addition, inspection seeks to complement testing [53], which can facilitate testing at a reduced cost and with a lighter testing schedule.

In summary, the benefits of inspection are significant. Industrial experiences have already shown that the use of software inspection can effectively improve software

[3] Variants Include Fagan Inspections [4], Active Design Reviews [57], N-fold Inspections [37], Phased Inspections [31], and Two-Person Inspections [12].

quality, provided that a rational and systematic procedure supported by sound effective tools is followed [3][53].

4.1.2 What We Mean by Inspection

Software inspection is a form of review process [7]. There is so far no concrete, universal agreement on the definition of an inspection process. In this work, by inspection we mean a systematic approach to examining a program in great detail a little at a time [53], i.e. evaluating the quality of the software in question and demonstrating that a program performs as desired. The subject of the inspection is the software, not the developing process of the software.

In general, inspection means assessing a product by following a prescribed, systematic process that aims to determine whether the product is fit for its purpose [53]. Fagan's inspection process [23] includes six well-defined steps (planning, preparation, meeting, rework, and follow-up), and the involvement of specific inspection roles (moderator, recorder, reader, and producer). Variations of Fagan's inspection process originate from the different tradeoffs between minimum interval, minimum effort, and maximum effectiveness: they are presented by restructuring Fagan's process, for instance, rearranging the steps, changing the number of people working on each step, or the number of times each step is executed (Some of the variants may require special detection methods) [7][58]. Documents are commonly used to ensure that the inspection is rigorous and complete.

Usually, inspectors need to not only follow a prescribed, systematic procedure, but also understand the product and the underlying technologies. This helps optimize and organize the proceedings of the inspection. However, since the participants' ability to handle details and complexity differs, a successful inspection should essentially rely more on a sound and systematic procedure when the task is large and complex. Appropriate tool assistance is also important.

Parnas et al. suggested that a policy of "divide and conquer" can be the key to such cases [53][57]. During inspection, the participants are required to examine small parts of the product in isolation rather than look at the product as a whole. However, in this process, no possible case is overlooked, and the whole product's correctness can be implied by the inspected components' correctness. Detail presentation of this policy is shown in the following sections.

4.1.3 Now and Future

Now, software inspection is widely used and recognized as a cost-effective way of removing defects [7]. Many different inspection methods have been proposed to improve the procedural performance with distinct structural change, and the application of the methods varies (see Footnote 3). Extensive research in the last three decades has provided valuable knowledge and experience; for instance, significant progress has been made in areas such as the inspection process, reading techniques, defect estimation techniques, and computer automated-tool assistance [3].

It is worthy of particular mention that Parnas and Weiss' active design reviews [57] are multiple-session inspection procedures. Each inspection is divided into several mini-inspections or "phases." Parnas reemphasized that the policy of "divide and conquer" is a key to improving inspection performance [53]. In this work, we will show that the adoption of this policy can further assist the program analysis and thus, allow for a more effective and reliable inspection procedure.

Recent articles [4][8][22][53] offer insights into how software practitioners can improve their inspections' effectiveness and applicability today. Theoretical underpinnings of these suggestions [8][48][52][53] are being continuously provided to help gain further improvements in the future. For example, Dunsmore et al. [22] proposed techniques for refining the inspection for a particular object-oriented system setting, which demonstrates how customizing the inspection process to a particular task can bring about more benefits; Anderson et al. [4] describes a tool for

making inspection of complex software systems more manageable, which showed that computer-aided inspection has great potential to improve the inspection performance.

4.2 Inspection of Sequential Programs Using Tabular Expressions

4.2.1 Overview

Parnas et al. have developed a more feasible, systematic, and rigorous inspection approach in an industrial setting [48], which combines the ideas of Fagan, Harlan Mills [38][39] and Parnas and his coworkers [50][52]. This approach, built on a foundation of mathematical relations that describe program behavior, uses tabular expressions that help the inspection to be more feasible and systematic [48].

As illustrated in chapter 2, tables are well suited for communicating detailed design information. They help to organize the information and present it for ease of use and management, using a structured format. Moreover, they can facilitate analysis of aspects such as completeness and consistency. Use of tabular expression helps to avoid exhaustive inspection by making all possible cases explicit, and by allowing checks that assure consideration of all possibilities, without overlapping cases.

4.2.2 "Divide and Conquer" Policy

In inspecting a complex program, application of an appropriate policy increases the effectiveness and lowers the reliance on the performer's ability. Active Design Review inspection [57] offered an application of the policy of "divide and conquer" and showed its effectiveness. Parnas et al. successfully applied an approach guided

by this policy in the inspection of safety critical control programs [48] (see Section 4.2.3).

Parnas et al. reemphasized [53] the application of this policy to the inspection process when describing the inspection's role in software quality assurance. The major idea of the policy is that the whole product is inspected by looking at its small components. The inspection process needs to be organized as a set of steps, each of which is simple enough to be carried out reliably and independently, but with the additional property that the completion of all the steps results in the completion of the task [53].

Tabular expressions have proven helpful in applying this policy [44][48] with a table form, which prevents possible overlooking problems in the inspection process. In addition, the structure of tables allows the reviewer to take breaks at the end of any entry's consideration, and to return to the right place any time after the break; this is especially useful when a task stretches over long time.

4.2.3 Software Inspection Based on Program-Function Tables

A software inspection approach based on a program-function table has been developed and applied to an inspection project of the software used in a nuclear plant that is currently in operation in Darlington, Ontario, Canada [48].

In this inspection approach, a tabular representation of the requirements and a mathematical description of what the program actually does are produced and compared to show whether or not they are equivalent. Any disparity reveals one or more defects in either the program or the requirements, or both. The core of this method is to precisely summarize the effects of the intended behavior of the program, by deriving a set of program-function tables from it. A detailed discussion of this process can be found in [48][51][56].

Mills [38][39] showed that conventional mathematics could be used to prepare precise descriptions of a program's behavior. Each program, describing one or more sequences of state changes for a given initial state [47], can be described by a function or relation that maps from the start states to the final states. Tabular notation presents information in an intuitive manner, thus improving the readability of the program's behavior descriptions, and making serious errors easier to spot [30][48][50].

Description of the program in the inspection process is based on a hierarchical decomposition of the program, which allows the program to be examined in small parts. Use of tabular expression enables the inspection and comparison to proceed systematically on a case-by-case basis, without looking at other interacting parts. This is a result of the application of the "divide and conquer" policy.

The inspection method presented by Parnas et al. [48] improves upon the original by structuring the tables into a set of displays [52]. Each of the displays is complete in itself and can be reviewed and inspected without considering the others. Its application in industry has shown that this approach can be a valuable tool in facilitating program development.

Experience shows that the inspection method based on the program-function table is a practically effective way of analyzing the expected behavior of a program and ensuring its quality. It is also believed that support with the appropriate tools [1][53] is required to reduce the cost and enhance the applicability of the inspection process.

4.3 Inspections of Concurrent Programs

4.3.1 Overview of Inspections of Concurrent Programs

In practice, it is extremely difficult to show whether or not a program with concurrent activities meets its requirements. The design of concurrent programs is subtle due to the interaction among different processes. There are many facts that can be deduced from each program statement, which can be arranged in many ways; as a result, one could easily overlook something important. Because of the inherent non-deterministic feature of concurrent programs, there is still a need for substantial progress in the application of inspection techniques.

In concurrency literature, discussions of typical inspection techniques of concurrent software are compared with those of formal specification of concurrent software and verification of the consistency between concurrent programs and their specifications. Many techniques[4] are available for performing a formal mathematical proof of consistency between the specification and a sequential program. Some techniques have been extended to ensure the quality of concurrent programs, for instance, the Owicki-Gries extension of Hoare logic [41], the Lamport extension of Hoare logic [33], and the Lamport extension of Dijkstra's weakest precondition logic [35]. However, the cost has proven too high for these techniques to be used practically in software development [10].

Model checking is a verification technique for a class of systems that can be modeled by a finite automaton [9]. By representing a program (or a system) to an automaton, and by representing a property to a logical formula, a model checking algorithm shows that whether the given model satisfies the property. This method can be used to verify a concurrent system, however, the algorithm usually only deals with finite state cases. Our approach presented in this thesis has no such limitation.

[4] Formal models such as NDISM [10] (for concurrent programs), Petri-Net, Axiomatic systems based on Hoare's logic, or Dijkstra's weakest precondition logic, temporial logic, etc.

Most importantly, in model checking approach the inspectors have to build a model and work on it for verification. However, in this work we checked the program directly and did not need to construct a special model.

4.3.2 An Inspection Approach Using Tabular Notation for Concurrent Programs

In this work, we are trying to present a sound and systematic approach using Tabular Expression to inspect concurrent programs. This method checks whether the concurrent program behaves as it intends to do, by looking at its requirement specification.

We must prove two types of program properties in the process:

1) Invariant property

This property ensures that the requirement predicate expressions, including a set of program invariant clauses, hold in all states of the program.

2) Liveness property

This property is defined by a theorem; thus, the program will not stop unless all processes have finished.

A brief description of the approach is given as follows:

1) Add auxiliary variables to the original program to make quantities of interest explicit, which will not affect the original behavior of the program, but will record extra information needed for the proof. This additional information must be included in the hidden state of the supporting mechanism for the processes but is not visible in the program text. The "control" pointer or instruction pointer is an example of this hidden information as it is not discernable directly from one of the program variables mentioned in the code.
2) Formulate the requirements as a mathematical specification, including appropriate program invariants that capture essential properties of the

execution. The invariant set must be identified that is sufficiently strong in order to gain an efficient inspection on the program.

3) Specify primitive operators in the program, such as synchronization primitives (this should have been done before the program was written.)
4) Rewrite the program so that each primitive statement in the program has a label, and make the transfer of control from statement to statement explicit by assigning a label value to a control variable (program instruction counter) for each statement in each process. The execution of each statement is conditioned depending on the value of the label.
5) Describe the resulting program by a tabular representation, program-function table. When preparing the table, it is important to capture the states fully in the variables. This ensures that the table completely describes the behavior of the program. A theorem that the table is complete and consistent is required to be proved.
6) Inspect the description column by column to show that every action maintains the invariant, and that the invariant plus some information about the final state, implies the desired goal with the use of predicate logic.

A detailed description of applying the approach to a concurrent program is given in the following chapters.

Chapter 5

An Example Application

A number of well-known examples of synchronization problems have been used extensively in the literature. This thesis takes one of the most famous problems, the "Readers/Writers problem" first formulated by Parnas, Courtois, and Heymans [15], as an example to demonstrate the proposed approach.

5.1 The Readers/Writers Problem

The Readers/Writers problem [15] introduces two types of processes: readers and writers. Both need to access a shared common resource such as a buffer, a database, etc. Readers may share the resource concurrently with an unlimited number of other readers, while writers, permitted to modify the resource, must have exclusive access.

In order to prevent the indefinite exclusion of readers or writers or both, a fairness policy must be enforced. Due to different fairness requirements, there are several variants of the original Readers/Writers problem. The original paper [15] presented two solutions, giving readers and writers different access priorities, with the use of a semaphore synchronization primitive.

The first solution gives readers priority over writers. By such a policy, no reader should wait to access the resource because a writer is waiting. That is, no reader will be made to wait unless a writer has already obtained the permission to access the shared resource. However, this policy can result in the starvation[5] of writers due to the continuously coming readers.

The second solution gives priority to writers. By such a policy, once a writer is ready to write, the writing will be performed as soon as possible. That is, when a writer arrives, only those readers who have already obtained permission to read are allowed to finish the execution; when a reader arrives behind a writer, the reader's access will be postponed until the writing is complete. In this case, it is possible for readers to starve due to a continuous stream of arriving writers.

We choose the first solution to demonstrate the application process of the proposed approach step by step.

5.2 The Original Program

Below is the original program for the Readers/Writers problem, taken from [15]. It is the solution to acquiring minimum delay for the readers, in which "no reader should wait simply because a writer is waiting for other readers to finish [15]."

Note that, the sample program is small but the problem is actually quite tricky and the solution was subtle, which was (at the time) the result of a lot of discussion and generated a lot of discussion after it was published [15].

[5] Starvation: a low-priority process never gets access to the processor due to the higher-priority access of other processes.

```
READER:
    P (mutex);
    rdcnt := rdcnt+1;
    if rdcnt=1 then P(w);
    V(mutex);
    READ;
    P(mutex);
    rdcnt := rdcnt-1;
    if rdcnt=0 then V(w);
    V(mutex);

WRITER:
    P(w);
    WRITE;
    V(w);
```

Figure 5-1 The Original Program

Variable **rdcnt** is used as a counter for all appearing readers, including those waiting to enter the critical section [6]. Two semaphores are used in the solution as synchronization primitives. The following sections are detailed explanations for the use of the semaphore synchronization mechanism.

5.2.1 Semaphore w

Semaphore w is used as a mutual exclusion semaphore for each writer process, which is only used for the first reader process entering the critical section (we name this critical section the w-critical-section for following use) and the last reader process leaving the critical section. This is because writers need exclusive access, while readers can access simultaneously. Semaphore w uses two primitive operators:

[6] Critical sections are the sections of code that use the shared resource such as the buffer, data base, storage unit, etc.

P(w) and V(w). For an initial value of "1", the two primitive operators of **w** will be invoked either by a writer or by the first/last reader.

Each P(w) operation decreases the counter of **w**, described as **w.cnt** (see the definition in Section 3.2.2), by one. The P(w) operation cannot be passed if the resulting value of **w** is negative. A successfully passed P(w) operation results in the executing process entering the critical section. If a reader has already been in the critical section, the subsequent readers will directly enter the section without operating on P(w). If the subsequent processes are writers, then their executions will be suspended and booked in the waiting list of **w**. When the value of **w** is non-positive, its absolute value equals the number of processes booked on **w**'s waiting list.

5.2.2 Semaphore mutex

The semaphore **mutex** is used only for readers' operations. Because more than one reader may request access to the critical section (w-critical-section) and execute on the shared variable **rdcnt** simultaneously, it is necessary to ensure that only one reader is operating on the variable **rdcnt** at a time.

Note that there are two classes of critical sections in the reader program: the original critical section, named w-critical-section in Section 5.2.1, where the processes read and write the shared data structure that motivates the whole problem; there is another class of critical section, which is protected by mutex and is named mutex-critical-section. The mutex-critical-section is used to indicate that only one reader can enter or leave the w-critical-section at a time. There are two mutex-critical-sections in the program, one for entering the read/write area, one for leaving.

Semaphore **mutex** has similar properties as semaphore **w**, thus a detailed understanding of the operation on **mutex** may be gained from the discussions in Section 5.2.1.

A mathematically rigorous description of the semaphore variable and its P-operation/V-operation in tabular representation are illustrated in Section 6.4.3.

5.3 Requirements of the Original Program

Considering the requirements of the Readers/Writers problem, we know that in the critical section [15]:

1) Only one writer can be writing while no reader is reading.
2) More than one reader can be reading concurrently while no writer is writing.

Let **rd** be the number of active[7] reader processes and **wt** the number of active writer processes. The above requirement can then be described by the following expression:

$$(rd=0 \vee wt=0) \wedge (wt<2)$$

Figure 5-2 The Requirement Expression

Initially, there are no readers or writers in the shared section of the programs. Therefore, the requirement expression is initially satisfied for rd=0 and wt=0. We only need to show that no action of any of the processes will make the above condition false if it is true before that action.

[7] active: indicates an execution status of a process; a process is active only when it is executing in the critical section and processing.

Chapter 6

Rewriting the Readers/Writers Program

6.1 Two Counter Variables

In the proposed approach, we first rewrite the program by adding two variables, **rd** and **wt**, which have been introduced in the aforementioned requirement expression (see figure 5-2). This step is essential, as the mathematical description of the requirements is given in terms of these two variables, and it is critical to note where the values of these variables can be changed.

The rewritten program with the two counter variables is given in figure 6-1 below:

```
READER:
    P(mutex);
    rdcnt := rdcnt+1;
    if rdcnt=1 then P(w);
    rd := rd+1;     {see the following note: a }
    V(mutex);
    READ;
    P(mutex);
    rdcnt := rdcnt-1;
    rd := rd-1;     {see the following note: b }
    if rdcnt=0 then V(w);
    V(mutex);

WRITER:
    P(w);
    wt := wt+1;
    WRITE;
    wt := wt-1;
    V(w);
```

Figure 6-1 The Program with Two Counter Variables

Note **a:** We do not consider a reader process as an active process until it passes the P-operation of semaphore **w** and enters the critical section, because readers cannot perform reading outside of the critical section. However, only the first reader needs to do the P-operation of semaphore **w**; the following readers can enter the critical section directly if there already have been readers reading according to the program requirement.

Note **b:** The decrement statement must be added at the location between the P/V operation of **mutex** as shown in the above program. If the statement is given earlier, before the **mutex** operation, we would have to worry about two processes executing it simultaneously and losing count of the number of readers. In fact, both **rdcnt** and

rd need protection in order to change their values within the section protected by **mutex**.

A similar explanation applies to the added statements in the writer process program.

6.2 When Can a Program Be Interrupted?

We assume that the component processes in a concurrent program are executed on some hardware or software platform. We can view the program execution in terms of state transitions, each of which we can consider primitive. A primitive statement is a statement that can be executed without interruption by other processes; once a process starts executing a primitive statement, other processes cannot intervene in this process. We need to introduce two symbols in order to apply our approach.

6.2.1 The "[ ]" Notation

We introduce a symbol "[ ]" to indicate that the statements inside the brackets are undivided and must be executed without any intervening events from other processes.

When the executing process enters the critical section after a P-operation, there must be an increment of the newly introduced counter variable **rd** or **wt,** since a newly activated process, either a reader or a writer, is added at the time. However, a V-operation will always lead to a decrement of the counter variable. In other words, the P-operation and its corresponding increment statement are indivisible; the same is true for the V-operation and the decrement statement.

Therefore, we put the P-operation and its corresponding increment statement in **"[ ]"** to indicate the indivisibility, and the V-operation and the decrement statement as well. We assumed that the sequence order of the P-operation/V-operation and the

increment/decrement statement of the counter variable does not matter when they are in the "**[ ]**".

Below is the rewritten program with symbol "[ ]":

```
READER:
        P(mutex);
        rdcnt := rdcnt+1;
        [ if rdcnt=1 then P(w); rd := rd+1;]
        V(mutex);
        READ;
        P(mutex);
        rdcnt := rdcnt-1;
        [if rdcnt=0 then V(w); rd := rd-1;]
        V(mutex);

WRITER:
        [ P(w); wt := wt+1;]
        WRITE;
        [V(w); wt := wt-1;]
```

Figure 6-2 The Rewritten Program with Symbol "[ ]"

Note that we are using the symbol "[ ]" only with the auxiliary variables that are added to help express the program requirements. We are not using it to combine statements that were in the original program.

6.2.2 The "stop" Symbol

The "**stop**" symbol tells us when a process under execution can be interrupted, allowing other processes to resume their execution.

Each line of the program in figure 6-2 is primitive, which allows it to be executed independently and it will terminate at the end of the line. That is, except for statements inside the "[ ]" notation, each statement stops at the semicolon, and the "**stop**" symbol can then appear at the point where a semicolon appears in the program.

The P-operation/V-operation statement and its corresponding increment/decrement statement are coupled in sequence. This is done by placing a "**stop**" symbol after each "[ ]". We found that after putting the "stop" statements in place, we can omit the "[ ]" symbol for simplicity.

The following program in figure 6-3 demonstrates this symbol's use.

```
READER:
        P(mutex)                          stop
        rdcnt := rdcnt+1                  stop
        if rdcnt=1 then P(w); rd := rd+1  stop
        V(mutex)                          stop
        READ                              stop
        P(mutex)                          stop
        rdcnt := rdcnt-1                  stop
        if rdcnt=0 then V(w); rd := rd-1  stop
        V(mutex)                          stop

WRITER:
        P(w); wt := wt+1                  stop
        WRITE                             stop
        V(w); wt := wt-1                  stop
```

Figure 6-3 The Rewritten Program with Symbol "stop"

Note that in this example the "**stop**" symbols are redundant, because semicolons simply indicate where they appear. However, there could be other situations in

which an interruption could occur between semicolons, or some of the semicolons would not be at the points where interruptions might occur.

6.3 Label the Primitive Statements

Primitive statements can be executed independently without interruption. Each statement execution results in an indivisible state change. The execution of a concurrent program can be considered as a sequence of executions of all primitive statements in all processes involved, in a non-deterministic manner. When several processes are ready to be executed, the choice of which process goes next is non-deterministic while the choice of which statement within a process will be executed next is deterministic. Based on such concerns, we can simply associate a control variable with each constituent process in the concurrent program. This variable is used to locate the primitive statement in the process that is to be executed next. The state of the program is defined by the values of the program variables as shown in Section 6.6.

The array variable **next** functions as an instruction counter variable. The index of the array is the identification integer of each constituent process of the program. The possible values of the array indicate the labels of the statements in the process. For example, the value of the variable **next[i]**, will be used in the $\mathbf{i}^{th}$ process, either a reader or a writer, to indicate the current label of the statement in process **i**.

We must follow three steps in the rewriting:

1) The values of the array variable **next** will function as labels to identify each primitive statement in the program: we use [r1...r9] and [w1...w3] to label each statement in the reader process and in the writer process, respectively.

2) We introduce "**if ... then ...**" to make the execution of the statement conditional on the value of the **next** variable to indicate the "instruction counter" for each process.

3) We assign values to the elements of the control array variable **next** to explicitly transfer the control from statement to statement. This is illustrated in figure 6-4 below.

In this way, each primitive statement in the program is given a label, the execution of each statement is conditional, and transfer of control from statement to statement is explicit. Any statement in the rewritten program can be executed at any time, but will only have an effect when its condition is met. When several statements in different processes are ready to proceed, we assume that the selection from the ready process list is random, but that a statement will not be executed unless there has been some state change since its most recent execution.

We present an illustration of the program for the **i**th reader process and the **j**th writer process shown as "**Reader i**" and "**Writer j**" in figure 6-4. We use array **next[i]**/ **next[j]** as the control variable for the two processes: process **i** and process **j**. Now an integer variable **pID** is introduced to indicate the identification of the current executing process; when the **i**th process is executed, **pID** equals **i**, and when the **j**th process is executed, **pID** equals **j** (see also Section 6.6).

Also note that there is no assignment statement to the control variable **next[i]**/ **next[j]** when a P-operation/V-operation appears in the program. For instance, there is no assignment statement such as "**next[i]:= r2**" in the 2nd line[8] of the **i**th reader process (see figure 6-4); there is no assignment statement such as "**next[j]:= w2**" in the 2nd line of the **j**th writer process (see figure 6-4). This is because we include the assignment statement as part of each P-operation/V-operation of the semaphore **w**. Detailed explanation can be found in the definition of the semaphore operations in Section 6.4.

Two symbols "**Begin**" and "**End**" are introduced to indicate the range of a reader or writer program segment, as shown in 1st line of the reader/writer program and in 11th line of the reader program, in 5th line of the writer program in figure 6-4. We confine our analyses of the program only in between "**Begin**" and "**End**".

[8] We number the statement line in the program only for discussion purpose and thereafter.

READER i:

```
1    Begin
2    if next[i]=r1  then  P(mutex)                                  stop
3    if next[i]=r2  then  rdcnt := rdcnt+1;     next[i]:=r3         stop
4    if next[i]=r3  then  if rdcnt=1 then P(w) ; rd := rd+1         stop
5    if next[i]=r4  then  V(mutex)                                  stop
6    if next[i]=r5  then  READ;                 next[i]:=r6         stop
7    if next[i]=r6  then  P(mutex)                                  stop
8    if next[i]=r7  then  rdcnt := rdcnt-1;     next[i]:=r8         stop
9    if next[i]=r8  then  if rdcnt=0 then V(w); rd := rd-1          stop
10   if next[i]=r9  then  V(mutex)                                  stop
11   End
```

WRITER j:

```
1    Begin
2    if next[j]=w1  then  P(w); wt := wt+1                          stop
3    if next[j]=w2  then  WRITE;                next[j]:=w3         stop
4    if next[j]=w3  then  V(w); wt := wt-1                          stop
5    End
```

Figure 6-4 The Rewritten Program with Labels

6.4 Specify Synchronization Primitive Operators

In Section 5.2 we explained how the two semaphores **w** and **mutex** work in the original program. In this section we first specify a mathematical description of a semaphore's P/V-operation and then explain how to rewrite the program, with a focus on the descriptions of the semaphore's operation.

6.4.1 The Execution States of Semaphore's P/V-operation

When discussing the P-operation of the semaphore (written as P(sem) below), we should consider three resulting possibilities:

a) The process passes P(sem) successfully and advances its execution;

b) The process is suspended and booked on the waiting list of the semaphore; or

c) The process is invoked from a suspended P(sem) and released from the waiting list.

In case a), there is no special treatment.

In case b), we introduced a special label value of array variable **next**, **"waitAtsem"**, to indicate a state in which a process is blocked by a P-operation and added to the semaphore ***waiting*** list to be waken up by a V-operation.

In case c), we introduced a special label value of array variable **next**, **"rlseAtsem"**, to indicate a state in which a process in a semaphore waiting list is activated and ***released*** by a V-operation and ready to advance its execution.

6.4.2 The Initialization of the Semaphore Variable

According to the description of a semaphore variable in Section 3.2.2, we use the following illustration as a definition of a semaphore variable:

```
Typedef struct{
        int: cnt;
        sset: set;
        } semaphore
```

where sset is a type of SET defined in Appendix A.

The initialization of a semaphore variable **sem**:

sem.cnt = 1;

sem.set = Φ;

6.4.3 The Tabular Representation of the P/V-operation

In this section, we first define a function NextLabel(x) that operates on a value x of the instruction counter variable, array **next**, then give the definition of the semaphore's P/V operation. The function is not part of the definition but is tailored to this program.

Function **NextLabel(x)** is defined to acquire the next value of the label, which is an element of variable **next** for a given label value of **x** (also an element of array **next**). The function is defined in figure 6-5.

Figure 6-5 The Definition of Function **NextLabel(x)**

X	NextLabel(x)
r1	r2
waitAtPm1	waitAtPm1
rlseAtPm1	r2
r2	r3
r3	r4
waitAtPwr	waitAtPwr
rlseAtPwr	r4
r4	r5
r5	r6
r6	r7
waitAtPm2	waitAtPm2
rlseAtPm2	r7
r7	r8
r8	r9
r9	END
w1	w2
waitAtPww	waitAtPww
rlseAtPww	w2
w2	w3
w3	END

The tabular representation of the P-operation:

Let **x** be the label of the currently executing statement of process **i**, Figure 6-6 describes the P-operation on semaphore **sem**, **P(sem)**, executed by process **i** where $i \in [1...N]$; and j is the index of array **next** (see its description in Section 6.6), where $j \in [1...N]$:

Figure 6-6 The Description of P-operation **P(sem)**

	'sem.cnt>1	'sem.cnt=1	'sem.cnt <1
sem.cnt'|	false	sem.cnt'= 'sem.cnt –1	sem.cnt'= 'sem.cnt -1
sem.set'|	false	sem.set' ='sem.set	sem.set' ='sem.set ∪{i}
next'|	false	tab 6-6-1	tab 6-6-2

tab 6-6-1:

$\forall j, j \in [1...N] \wedge$

j=i	j≠i
next[j]'=NextLabel(x)	next[j]'= 'next[j]

tab 6-6-2:

$\forall j, j \in [1...N] \wedge$

j=i	j≠i
next[j]'=waitAtPsem	next[j]'= 'next[j]

The tabular representation of the V-operation:

Let **x** be the label of the currently executing statement of process **i**, Figure 6-7 describes the V-operation on semaphore **sem**, **V(sem)**, executed by process **i** where $i \in [1...N]$; and j is the index of array **next** (see its description in Section 6.6), where $j \in [1...N]$:

Figure 6-7 The Description of V-operation **V(sem)**

	'sem.cnt>0	'sem.cnt =0	'sem.cnt <0
sem.cnt'|	false	sem.cnt'='sem.cnt +1	'sem.cnt +1
sem.set'|	false	sem.set'= 'sem.set	(∃t, (sem.set'= ('sem.set-{t}) ∧ t∈'sem.set ∧ t∉sem.set'))
next'|	false	tab 6-7-1	tab 6-7-2

tab 6-7-1:

∀j, j∈[1...N] ∧

j=i	j≠i
next[j]'=NextLabel(x)	next[j]'= 'next[j]

tab 6-7-2:

∀j, j∈[1...N] ∧

j=i	(j≠i) ∧ (j∈('sem.set-sem.set')) ∧ 'next[j]=waitAtPsem	(j≠i) ∧ (j∉('sem.set-sem.set'))
next[j]'=NextLabel(x)	next[j]'=rlseAtPsem	next[j]'= 'next[j]

In section 6.4.1 we described that a process may be suspended in a P-operation and released by another process's V-operation from the waiting list. By **'sem.cnt** in figure 6-6 and 6-7 we mean the value of the semaphore just before each process starts the P/V-operation. There is always a value change of **sem.cnt** in a P-operation whether or not the process will pass or be suspended after the operaion. A waiting process, when released, will not go back to recheck **sem.cnt** and change the value but advance to its next statement with label of relsAtPsem. And sem.cnt' is the value after the process did the decrement not after they have been released from the waiting state by another process. The value of sem.cnt after the P-operation is completed is determined by the V-operation that released the waiting process.

6.4.4 Explanations of the Semaphores in the Program

Both w and mutex can have the three possibilities discussed in Section 6.4.1.

(1) For semaphore **w**, we have the following implementation:

For case a) in Section 6.4.1, we bind the P(w) operation with the increment statement of the counter variables **rd** or **wt**, as illustrated before in figure 6-4.

For case b) in Section 6.4.1, we have two labels, "**waitAtPwr**" for reader processes and "**waitAtPww**" for writer processes. When the executing process is blocked at the P-operation, no further advancement of the execution is made at the time, and the array variable **next** is assigned to a value of **waitAtPwr** or **waitAtPww.**

Note that when one of the statement labels, **waitAtPwr** or **waitAtPww,** in the statement condition is met, no execution is actually performed; that is, the control variable **next** of the process is re-assigned to the same value: **waitAtPwr** or **waitAtPww.** This is because a waiting process can only be activated through a V-operation performed by another process; the waiting process will stay suspended when there is no V-operation invoking its advancement. A sample illustration can be found in statement line 3 of the rewritten reader program in figure 6-8.

For case c) in Section 6.4.1, we introduce two labels "**rlseAtPwr**" for reader processes and "**rlseAtPww**" for writer processes. When the waiting process, either a reader or a writer, is re-activated by the V-operation of another process, an appropriate label, either **rlseAtPwr** or **rlseAtPww,** will be assigned to the waiting process to indicate that it is released and ready to advance; the waken-up is performed by the another process' V-operation. This is specified in the definition of the semaphore's V-operation in Section 6.4.3. In this situation, the process will advance with an increment of the counter variable **rd** or **wt**, which is illustrated in figure 6-8(see statement line 8 of reader program and line 4 of writer program).

When V(w) is completed, the executing writer or the last reader will leave the w-critical-section and release the access permission to a process in the waiting list of **w,** which is then activated to resume its execution. If the value of **w** resulting from the recent V(w) operation is negative, then the waiting list of **w** would contain more than one process. In such a case, more than one process can be permitted to advance. We assume that whether or not the waiting process can be removed from the waiting list remains undefined and the selection is non-deterministic.

(2) Semaphore **mutex** has similar properties as semaphore **w.** For semaphore mutex we need to consider two mutex-critical-sections and only with reader processes (see explanation in Section 5.2.2).

We introduced two label values, **waitAtPm1** or **waitAtPm2,** for the two mutex-critical-sections respectively. When a **P(mutex)** operation is suspended, the control variable **next** of the executing process will be assigned to a value of **waitAtPm1** or **waitAtPm2**; accordingly, when the waiting reader process is invoked, the variable **next** will be assigned to a value of **rlseAtPm1** or **rlseAtPm2.** Pertinent explanation of the operation on **mutex** may be gained from the above discussions.

The resulting program is demonstrated in figure 6-8 in the following section.

6.5 The Rewritten Program

READER i:

```
1   Begin
2   if next[i]=r1          then  P(mutex)                              stop
3   if next[i]=waitAtPm1   then                 next[i]:= waitAtPm1    stop
4   if next[i]=rlseAtPm1   then                 next[i]:=r2            stop
5   if next[i]=r2          then  rdcnt := rdcnt+1; next[i]:=r3          stop
6   if next[i]=r3          then  if rdcnt=1 then  P(w) ; rd := rd+1     stop
7   if next[i]=waitAtPwr   then                 next[i]:=waitAtPwr     stop
8   if next[i]=rlseAtPwr   then  rd := rd+1;    next[i]:=r4            stop
9   if next[i]=r4          then  V(mutex)                              stop
10  if next[i]=r5          then  READ;          next[i]:=r6            stop
11  if next[i]=r6          then  P(mutex)                              stop
12  if next[i]=waitAtPm2   then                 next[i]:=waitAtPm2     stop
13  if next[i]=rlseAtPm2   then                 next[i]:=r7            stop
14  if next[i]=r7          then  rdcnt := rdcnt-1; next[i]:=r8          stop
15  if next[i]=r8          then  if rdcnt=0 then V(w); rd := rd-1       stop
16  if next[i]=r9          then  V(mutex)                              stop
17  End
```

WRITER j:

```
1   Begin
2   if next[j]=w1          then  P(w); wt := wt+1                      stop
3   if next[j]=waitAtPww   then          next[j]:=waitAtPww            stop
4   if next[j]=rlseAtPww   then  wt := wt+1;  next[j]:=w2              stop
5   if next[j]=w2          then  WRITE;       next[j]:=w3              stop
6   if next[j]=w3          then  V(w); wt := wt-1                      stop
7   End
```

Figure 6-8 The Resulting Program

Note that, in figure 6-8,

1) In line 2, 11 of the reader program, we can replace **P(sem)** with **P(mutex)** in figure 6-6;
2) In line 6 of the reader program and line 2 of the writer program, replace **P(sem)** with **P(w)** in figure 6-6;
3) In line 9,16 of the reader program, replace **V(sem)** with **V(mutex)** in figure 6-7;
4) In line 15 of the reader program and line 6 of the writer program, replace **V(sem)** with **V(w)** in figure 6-7.

Proof of Behavioral Equivalence:

The rewritten program has the same behavior of the original program, although it contains statements and variables that do not appear in the original program. The additional variables, also called auxiliary variables, and the statements using them do not affect the flow of control and any of the existing variables in the original program. The newly added variables are used to keep track of the program and to observe the program state. In other words, if we had a trace of the new program in which we left out all of the new variables, we would find the same set of possible event sequences. Therefore, the analysis of the rewritten program and the original program will have the same result.

6.6 Representation of the Program State

The program state for the rewritten program can be described as a 7-tuple **(rdcnt, rd, wt, mutex, w, next, pID)**. From figure 6-8, every variable of the original state, including **rdcnt**, **mutex**, and **w**, is in the rewritten program's state and that the additional variables, including **rd**, **wt**, **next**, and **pID**, do not alter the sequence of events of the program. Use of the auxiliary variables is only to record progress explicitly rather than implicitly.

The following is a detailed explanation of the use of the variables:

rdcnt is the counter of all readers already entered or still waiting to enter the critical section.

type: int
initial value: 0
possible value: 0,1,2,...MaxReaderNumber, where MaxReaderNumber describes the possible maximum number of the readers in the system.

rd is the counter of all active[9] readers in the w-critical-section.

type: int
initial value: 0
possible value: 0,1,2,...MaxReaderNumber

wt is the counter of all active writers in the w-critical-section

type: int
initial value: 0
possible value: 0,1

Note that, there can only be one writer in the w-critical-section according to the program requirement.

mutex is a semaphore variable in the program that functions as a mutual exclusion semaphore for readers to ensure that only one reader will enter or leave the w-critical-section at a time, in other words, there is only one reader working on the shared variable **rdcnt** in the two mutex-critical-sections.

type: semaphore
initial value: mutex.cnt=1;
mutex.set=∅;

The type and possible values of mutex.cnt and mutex.set are listed in figure 6-9 below:

[9] Active means the reader/writer process is actively reading/writing in the w-critical-section , but not waiting outside the section.

Name	Type	Possible Values
mutex.cnt	int	-(MaxReaderNumber-1), ... , -1, 0, 1
mutex.set	set of int	$\mathcal{P}(S)$[10], S⊂$\mathcal{N}$, where $\mathcal{N}$ is the set of non-negative integers

Figure 6-9

According to the definition of the semaphore operations, when there is one process waiting on the semaphore variable, the counter of the semaphore will be decreased by one. Therefore, in figure 6-9 when **mutex.cnt** equals (**MaxReaderNumber -1**), it corresponds to a state in which a reader is executing at label **r1** (in line 2 of reader process in figure 6-8) while all other readers are waiting to pass **P(mutex)**.

w is the semaphore variable used to provide mutual exclusion in the w-critical-section shared by readers and writers in the program.

type: semaphore
initial value: w.cnt=1;
w.set=∅;

The type and possible values of w.cnt and w.set are listed in figure 6-10 below:

Name	Type	Possible Values
w.cnt	int	-MaxWriterNumber, ... , -1, 0, 1
w.set	set of int	$\mathcal{P}(S)$, S⊂$\mathcal{N}$, where $\mathcal{N}$ is the set of non-negative integers

Figure 6-10

MaxWriterNumber in figure 6-10 describes the possible maximum number of writers in the system. When **w.cnt** equals **MaxWriterNumber,** it indicates a states

[10] $\mathcal{P}(S)$ denotes the power set of set S.

in which a writer is executing at label **w1** (in line 2 of writer process in figure 6-8) while all other writers are waiting to pass P(w) and at same time a reader comes to label **r3** (in line 6 of reader process in figure 6-8) and tries to pass P(w); or that a reader is executing at label **r3** while all writers come to label **w1** and wait to pass P(w).

next is the control variable for each process. The elements of array **next** are used to specify the next executing statement in a reader or writer process. The indices of the array are identifiers of the processes.

The indices of array **next** are a contiguous set of integers beginning with 1. The array consists of two parts: the element values in the front part represent the identities of all the reader processes, while those in the end part account for all the writer processes.

type: array

initial value: **r1** for reader process and **w1** for writer process[11]

possible value: r1, r2,..., r9, waitAtPwr, waitAtPm1, waitAtPm2, rlseAtPwr, rlseAtPm1, or rlseAtPm2 for readers, and w1,w2,w3, waitAtPww, or rlseAtPww for writers.

Note that there are two categories of the values, one for the reader processes and one for the writer processes. We assume that a value is assigned to an element of the array according to the type of the process either a reader or a writer process.

pID is an integer variable used to identify a chosen process in executing of the program (see also Section 6.3).

type: integer

possible value x, where $x \in N$ and N is the set of non-negative integers.

[11] A reader process and a writer process start the execution with different start states.

Chapter 7

The Tabular Representation of the Rewritten Program

The program-function table we construct in this chapter is the tabular representation of a single non-deterministic sequential program, which, as mentioned in chapter 3, is equivalent to the set of concurrent interacting processes (i.e. readers and writers) in the rewritten program (see figure 6-8).

The basic principle in constructing a program-function table is to capture the states fully in the variables, which ensures that the table completely describes the behavior of the program. This requires careful configuration of the table structure.

This chapter consists of three sections. The first section describes the abbreviation techniques that are necessary for construction of the program-function table; the second section explains in detail how to prepare such a table; the third section presents the resultant table (abbreviated) for the rewritten program, with an illustration of the actual table.

7.1 Explanation for the Abbreviation Techniques

More than one reader or writer process can be executed concurrently. All of these processes have to be described in the table to fully express the program behavior. If

we described the actual columns of all of the reader and writer processes, the resulting table would be very large with a copy of the representative columns for each process in execution.

To deal with this problem, we picked a generic table of each of the two classes of the process, readers and writers. We introduced a parameter **k** to denote the identification of a representative process. The use of this parameter in conjunction with all of the statements at the header of the table reduces the size of the table, leaving only a copy of the typical columns of one reader and one writer. This means that, through making statements about the parameter **k**, we are only looking at the representative set of columns of the real table. The tabular representation for the program is given only with the $\mathbf{k}^{th}$ ($k \in [1...M]$, where M is the number of readers and writers in total) reader/writer process. Note that when we look at the $\mathbf{k}^{th}$ process, either a reader or a writer, the process indicator **pID** equals **k** as discussed in Section 6.3.

We introduced two Boolean expressions **IsReader** and **IsWriter,** which stand for an abbreviation of $\mathbf{0<k\leq n}$ and $\mathbf{n<k\leq M}$ respectively, where $\mathbf{n \in M}$ (**n** is the number of reader processes, and **M** is the number of readers and writers in total). The two expressions indicate whether the current process is a reader or writer process. Also we use **IsReader** and **IsWriter** with parameter **k** in the header of the table to make the table consistent (or proper). This means that there will be no overlapping among the two representative sets of columns for readers and writers (see illustration in figure 7-4).

In order to make the tabular expression more readable and simple, we write "$\forall$j" to stand for "($\forall$j, $j \in [1...M]$) such that", where **M** is the number of all readers and writers in total. This is shown in the sub-tables for the variable **next** (figure 7-4).

7.2 How to Prepare the Program-function Table

As discussed in chapter 2, a program-function table is essentially a mixed-vector table, which has proven to be effective for describing a program's functional behavior [48][50]. The vector header at the left of the table indicates how the program variables with corresponding values in each row will be described; the normal header at the top of the table shows how many situations in question must be distinguished.

When constructing such a table, we need to go through the following two basic steps:

1) Configure the table structure:
In this step, we need to

(a) distinguish the column conditions according to the possible situations in the execution of each program statement;

(b) decide the variables that should appear in the vector header of the table, and choose the operator, either "equal to, =" or "such that, |".

(a)
The top header of the table should cover all of the situations that need to be distinguished when the program is executed. The Readers/Writers problem involves two types of processes, readers and writers; correspondingly, there would be two sets of representative columns appearing in the table. For each type of process, we describe each of the statements in one or more columns (see explanation below).

The program described in figure 6-8 consists of a set of "if-then" statements, each of which is primitive. Each primitive statement is described as at least one column in the table according to the statement precondition (predicate condition with before-notation variable names). Some statements describe more than one situation in execution, for example, in a P-operation on semaphore **mutex**, three situations

corresponding to the values of **mutex.cnt**[12] (>1, =1, and <1) need to be considered. In such cases, there should be more than one column in the table, describing the different possible situations for each of such statements. Therefore, in the above example, three columns with preconditions (‘**m.cnt**>1, ‘**m.cnt**=1, and ‘**m.cnt**<1) are included in the table.

When constructing the column header, we utilize the technique of "dotted lines" (see Section 2.3 for detailed description) to reduce visual clutter of common sub-expressions. In this way, the normal header of the table consists of multi-rows by the dotted lines; the actual column condition is the conjunction of the conditions in each row in the header.

(b)
The vector header of the program-function table covers all of the variables in the program that will have effect on a state change. Among them, the variable name such as **"rdcnt', rd' "** appearing in each row header uses an after-notation. The choice of operator, either "equal to, =" or "such that, |" (see explanation in Section 2.3) depends on whether or not a term or a predicate expression appears in the corresponding grid entry.

For Readers/Writers program, the variables in the vector header include shared variables in the program, such as **rdcnt, rd, wt, mutex** and **w**, and the instruction pointer variable **next[i]** that exists for each process **i**.

We describe the two semaphore variables **mutex** and **w** in the table according to the semaphore definition (see Section 6.4.4). Each semaphore variable includes two parts: semaphore.cnt and semaphore.set, therefore, two rows for each semaphore variable are added in the left header of the table: one describes the semaphore counter, **m.cnt** and **w.cnt**; the other describes the set of the waiting processes on the semaphore, **m.set**[13] and **w.set**.

[12] This is abbreviated as **m.cnt** in the following sections for simplicity.
[13] This stands for **mutex.set** here and thereafter.

The P/V-operation process may involve operations on **m.set** or **w.set.** This requires that the values of **m.set** and **w.set** be described as predicate expressions. Therefore, "such that, |" operators are used in those two row headers.

Variable **next** has sub-tabular expressions as cell values (see detailed explanation below), so it also needs "such that, |" operator in the row header.

Below is a sample transformation from statements to table.

Two sample statements (taken from figure 6-8) are provided in figure 7-1. The corresponding table structure after transformation is given in figure 7-2.

```
2     if next[i]=r1     then     P(mutex)                          stop
.....
5     if next[i]=r2     then     rdcnt:=rdcnt+1; next[i]:=r3       stop
.....
```

Figure 7-1 Sample Statements in Reader Process

	pID=k ∧ IsReader ∧					
	'next[k] = r1			.....	'next[k] = r2	.....
	'm.cnt>1	'm.cnt=1	'm.cnt<1			
rdcnt' =						
rd' =						
wt' =						
m.cnt'=						
m.set'|						
w.cnt' =						
w.set'|						
next'|						

Figure 7-2 The Corresponding Table Structure for Statements in Figure 7-1

Note that, the sample statement 2 (with line number 2) corresponds to the left three columns, while the sample statement 5 (with line number 5) corresponds to one column.

2) Determine the value for each table cell.
We determine the values of each variable in the row header according to the statement execution. In the above sample statements in figure 7-1, we find that:

For statement 2, because of the **P(mutex)** operation, the variables **m.cnt**, **m.set**, and **next[i]** may have been changed during the operation. There are three possible cases, which are discussed below:

Case 1: '**m.cnt=1**
There is no process in the mutex-critical-section (see Section 5.2.2) yet, so process **i** will pass **P(mutex)** successfully. According to the definition of P-operation (see Section 6.4.3), **m.cnt** will then be decreased by one and **next[k]** will be assigned with NextLabel(**r1**), i.e. **r2**.

Case 2: '**m.cnt<1**
The current process needs to wait in the semaphore list. According to the definition of P-operation, m.cnt will be decreased by one, i.e. **m.set'='m.set-1**; The identification of the current process **k** will be put into the waiting list of semaphore mutex, i.e. **m.set'='m.set**$\cup${**k**}; the value of variable **next[k]** will be assigned with the label value of waitAtPm1.

Case 3: '**m.cnt>1**
This will never occur since it contradicts the definition of semaphore: i.e. semaphore.cnt$\leq$1. In this case, we fill this column in gray.

Note that, for a statement with a V-operation on semaphore variables, the value of the semaphore will be increased by one, for example, **m.cnt'='m.cnt+1**. Also, the semaphore set, such as **m.set** and **w.set**, may have a change since a waiting process may be invoked to advance its execution, while the selection of the activated process

is random. In such cases, we use "**t**" to identify the randomly chosen process, thus, the after-value of the semaphore set can be:

(∃t, (semphore.set'= 'semphore.set-{t})).

About the row of the variable **next** in the table:

Each element of the array variable **next** is the instruction pointer for each process (see Section 6.6). This requires that we provide the value of each element in array **next** for each process instead of only the value of the $\mathbf{k}^{th}$ element in array **next** for the current process **k**. To do so, we distinguish two situations based on whether or not the process is the current process **k**. Therefore, when filling the values to the table cells of variable **next**, we need to define sub-columns with conditions of **i=k** or **i≠k**, where **i** denotes an element of array **next**, and **k**, the identification of the current process.

We use sub-table to express the values of such table cells of variable **next.** Also, we number each column in the table for convenience. For example, we use **tab2** to express the value of the cell of row **next** in column **2**.

For statement 5 in figure 7-1, there are only two assignments (**rdcnt':='rdcnt+1; next[i]:=r3**) executed when the statement condition is met, therefore, we only need to fill the grid cells corresponding to **rdcnt** and **next** with changed values.

The variables without changes in execution in both statement 2 and 5 will remain unchanged; the after-notation variable has same value of the before-notation variable, such as **rdcnt'= 'rdcnt**.

Figure 7-3 (a) is the resultant table with values from figure 7-2:

	pID=k ∧ IsReader ∧					
	'next[k] = r1			.....	'next[k] = r2	.....
	'm.cnt>1	'm.cnt=1	'm.cnt<1			
rdcnt' =		'rdcnt	'rdcnt		'rdcnt+1	
rd' =		'rd	'rd		'rd	
wt' =		'wt	'wt		'wt	
m.cnt'=		'm.cnt -1	'm.cnt -1		'm.cnt	
m.set'|		m.set' = 'm.set	m.set' = 'm.set∪{k}		m.set' = 'm.set	
w.cnt' =		'w.cnt	'w.cnt		'w.cnt	
w.set'|		w.set' = 'w.set	w.set' = 'w.set		w.set' = 'w.set	
next'|		Tab2	Tab3		Tab6	
	1	2	3	.....	6	.....

Figure 7-3 (a) The Resultant Table for Statements 2 and 5

Tab2 : ∀j,

j=k	j≠ k
next[j]'=NextLabel(r1)	next[j]'= 'next[j]

Tab3: ∀j,

j= k	j≠ k
Next[j]'=waitAtPm1	next[j]'='next[j]

Tab6: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r2)	next[j]'= 'next[j]

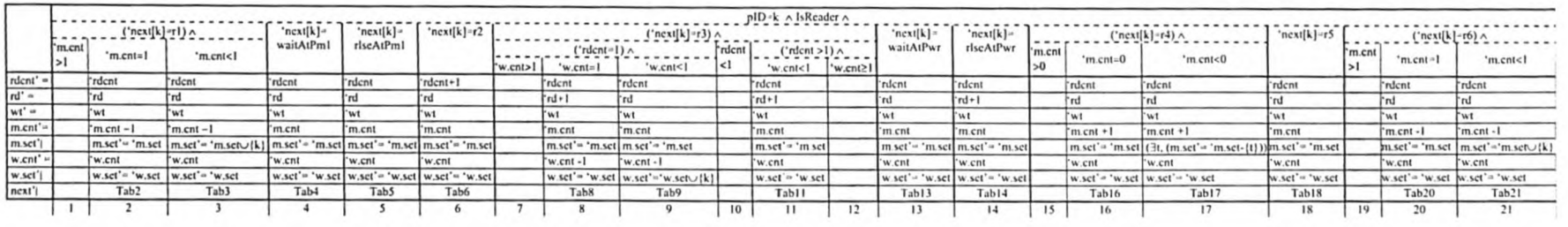
Figure 7-3 (b) Sub-tables 2, 3 and 6

7.3 The Tabular Representation of the Rewritten Program

7.3.1 The Tabular Representation of the Rewritten Program

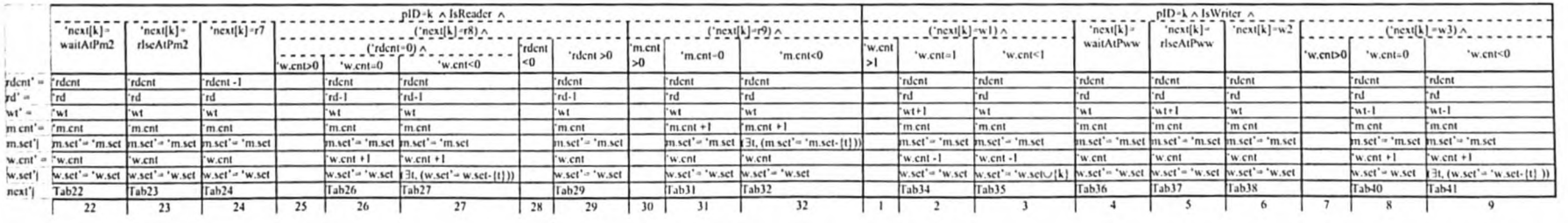
In the table shown in figure 7-4:
1) **m** is the abbreviation for **mutex** to reduce the width of the table.
2) **M** is the number of all reader and writer processes in execution.
3) **k** is the parameter introduced to indicate a typical executing reader and writer process.
4) **IsReader** is the abbreviation for $0<k\leq n$, where $n\in M$ and **n** is the number of reader processes.
5) **IsWriter** is the abbreviation for $n<k\leq M$, where $n\in[1..M]$ and **(M-n)** is the number of writer processes.
6) A gray column indicates that this case can never occur;
7) The values of array **next** in the table are described in sub-tables having a name format of "**Tab-col**", in which "**col**" is the present column number.

Figure 7-4 The Tabular Representation of the Rewritten Program



	pID=k ∧ IsReader ∧																						
	('next[k]=r1) ∧			'next[k]= waitAtPm1	'next[k]= rlseAtPm1	'next[k]=r2	('next[k]=r3) ∧						'next[k]= waitAtPwr	'next[k]= rlseAtPwr	('next[k]=r4) ∧			'next[k]=r5	('next[k]=r6) ∧		
	'm.cnt >1	'm.cnt=1	'm.cnt<1				'w.cnt>1	('rdcnt=1) ∧ 'w.cnt=1	('rdcnt=1) ∧ 'w.cnt<1	'rdcnt <1	('rdcnt >1) ∧ 'w.cnt<1	('rdcnt >1) ∧ 'w.cnt≥1			'm.cnt >0	'm.cnt=0	'm.cnt<0		'm.cnt >1	'm.cnt=1	'm.cnt<1
'rdcnt' =		'rdcnt	'rdcnt	'rdcnt	'rdcnt	'rdcnt+1		'rdcnt	'rdcnt		'rdcnt		'rdcnt	'rdcnt		'rdcnt	'rdcnt	'rdcnt		'rdcnt	'rdcnt
'rd' =		'rd	'rd	'rd	'rd	'rd		'rd+1	'rd		'rd+1		'rd	'rd+1		'rd	'rd	'rd		'rd	'rd
'wt' =		'wt	'wt	'wt	'wt	'wt		'wt	'wt		'wt		'wt	'wt		'wt	'wt	'wt		'wt	'wt
'm.cnt'=		'm.cnt −1	'm.cnt −1	'm.cnt	'm.cnt	'm.cnt		'm.cnt	'm.cnt		'm.cnt		'm.cnt	'm.cnt		'm.cnt +1	'm.cnt +1	'm.cnt		'm.cnt -1	'm.cnt -1
'm.set'		'm.set'= 'm.set	'm.set'= 'm.set∪{k}	'm.set'= 'm.set	'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set		'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set	(∃t, ('m.set'= 'm.set- {t}))	'm.set'= 'm.set		'm.set'= 'm.set	'm.set'='m.set∪{k}
'w.cnt' =		'w.cnt	'w.cnt	'w.cnt	'w.cnt	'w.cnt		'w.cnt -1	'w.cnt -1		'w.cnt		'w.cnt	'w.cnt		'w.cnt	'w.cnt	'w.cnt		'w.cnt	'w.cnt
'w.set'		'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= 'w.set	'w.set'='w.set∪{k}		'w.set'= 'w.set		'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= 'w.set	'w.set'= 'w.set
next'		Tab2	Tab3	Tab4	Tab5	Tab6		Tab8	Tab9		Tab11		Tab13	Tab14		Tab16	Tab17	Tab18		Tab20	Tab21
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Figure 7-4 Continued



	pID=k ∧ IsReader ∧												pID=k ∧ IsWriter ∧								
	'next[k]= waitAtPm2	'next[k]= rlseAtPm2	'next[k]=r7	('next[k]=r8) ∧ 'w.cnt>0	('next[k]=r8) ∧ ('rdcnt=0) ∧ 'w.cnt=0	('next[k]=r8) ∧ ('rdcnt=0) ∧ 'w.cnt<0	'rdcnt <0	'rdcnt >0	'm.cnt >0	('next[k]=r9) ∧ 'm.cnt=0	('next[k]=r9) ∧ 'm.cnt<0	'w.cnt >1	('next[k]=w1) ∧ 'w.cnt=1	('next[k]=w1) ∧ 'w.cnt<1	'next[k]= waitAtPww	'next[k]= rlseAtPww	'next[k]=w2	('next[k]=w3) ∧ 'w.cnt>0	('next[k]=w3) ∧ 'w.cnt=0	('next[k]=w3) ∧ 'w.cnt<0
'rdcnt' =	'rdcnt	'rdcnt	'rdcnt -1		'rdcnt	'rdcnt		'rdcnt		'rdcnt	'rdcnt		'rdcnt	'rdcnt	'rdcnt	'rdcnt	'rdcnt		'rdcnt	'rdcnt
'rd' =	'rd	'rd	'rd		'rd-1	'rd-1		'rd-1		'rd	'rd		'rd	'rd	'rd	'rd	'rd		'rd	'rd
'wt' =	'wt	'wt	'wt		'wt	'wt		'wt		'wt	'wt		'wt+1	'wt	'wt	'wt+1	'wt		'wt-1	'wt-1
'm.cnt'=	'm.cnt	'm.cnt	'm.cnt		'm.cnt	'm.cnt		'm.cnt		'm.cnt +1	'm.cnt +1		'm.cnt	'm.cnt	'm.cnt	'm.cnt	'm.cnt		'm.cnt	'm.cnt
'm.set'	'm.set'= 'm.set	'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set		'm.set'= 'm.set	(∃t, ('m.set'= 'm.set- {t}))		'm.set'= 'm.set	'm.set'= 'm.set	'm.set'= 'm.set	'm.set'= 'm.set	'm.set'= 'm.set		'm.set'= 'm.set	'm.set'= 'm.set
'w.cnt' =	'w.cnt	'w.cnt	'w.cnt		'w.cnt +1	'w.cnt +1		'w.cnt		'w.cnt	'w.cnt		'w.cnt -1	'w.cnt -1	'w.cnt	'w.cnt	'w.cnt		'w.cnt +1	'w.cnt +1
'w.set'	'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= 'w.set	(∃t, ('w.set'= w.set- {t}))		'w.set'= 'w.set		'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= 'w.set	'w.set'= 'w.set∪{k}	'w.set'= 'w.set	'w.set'= 'w.set	'w.set'= 'w.set		'w.set'= w.set	(∃t, ('w.set'= 'w.set- {t}))
next'	Tab22	Tab23	Tab24		Tab26	Tab27		Tab29		Tab31	Tab32		Tab34	Tab35	Tab36	Tab37	Tab38		Tab40	Tab41
	22	23	24	25	26	27	28	29	30	31	32	1	2	3	4	5	6	7	8	9

The following are the sub-tables for the table in figure 7-4:

Tab2 of Readers: ∀j,

j=k	j≠ k
next[j]'=NextLabel(r1)	next[j]'= 'next[j]

Tab3 of Readers: ∀j,

j= k	j≠ k
next[j]'=waitAtPm1	next[j]'='next[j]

Tab4 of Readers: ∀j,

j= k	j≠ k
next[j]'= 'next[j]	next[j]'= 'next[j]

Tab5 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(rlseAtPm1)	next[j]'= 'next[j]

Tab6 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r2)	next[j]'= 'next[j]

Tab8 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r3)	next[j]'= 'next[j]

Tab9 of Readers: ∀j,

j= k	j≠ k
next[j]'=waitAtPwr	next[j]'= 'next[j]

Tab11 of Readers : ∀j,

j= k	j≠ k
next[j]'=NextLabel(r3)	next[j]'= 'next[j]

Tab13 of Readers: ∀j,

j= k	j≠ k
next[j]'= 'next[j]	next[j]'= 'next[j]

Tab14 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(rlseAtPwr)	next[j]'= 'next[j]

Tab16 of Readers: ∀j,

j= k	j≠k
next[j]'=NextLabel(r4)	next[j]'= 'next[j]

Tab17 of Readers: ∀j,

j=k	j≠k ∧ (j∈('m.set-m.set')) ∧		j≠ k ∧
	'next[j]=waitAtPm1	'next[j]=waitAtPm2	(j∉('m.set-m.set'))
next[j]'=NextLabel(r4)	next[j]'=rlseAtPm1	next[j]'=rlseAtPm2	next[j]'= 'next[j]

Tab18 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r5)	next[j]'= 'next[j]

Tab20 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r6)	next[j]'= 'next[j]

Tab21 of Readers: ∀j,

j= k	j≠ k
next[j]'=waitAtPm2	next[j]'= 'next[j]

Tab22 of Readers: ∀j,

j= k	j≠ k
next[j]'= 'next[j]	next[j]'= 'next[j]

Tab23 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(rlseAtPm2)	next[j]'= 'next[j]

Tab24 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r7)	next[j]'= 'next[j]

Tab26 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r8)	next[j]'= 'next[j]

Tab27 of Readers: ∀j,

j= k	(j≠ k) ∧ (j∈('w.set-w.set')) ∧'next[j]=waitAtPww	(j≠ k) ∧ (j∉('w.set-w.set'))
next[j]'=NextLabel(r8)	next[j]'=rlseAtPww	next[j]'= 'next[j]

Tab29 of Readers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(r8)	next[j]'= 'next[j]

Tab31 of Readers: ∀j,

j= k	j≠ k
next[j]'=END	next[j]'= 'next[j]

Tab32 of Readers: ∀j,

j= k	(j≠ k) ∧ (j∈('m.set-m.set')) ∧		(j≠ k) ∧ (j∉('m.set-m.set'))
	'next[j]=waitAtPm1	'next[j]=waitAtPm2	
next[j]'=END	next[j]'=rlseAtPm1	next[j]'=rlseAtPm2	next[j]'= 'next[j]

Tab2 of Writers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(w1)	next[j]'= 'next[j]

Tab3 of Writers: ∀j,

j= k	j≠ k
next[j]'= waitAtPww	next[j]'= 'next[j]

Tab4 of Writers: ∀j,

j= k	j≠ k
next[j]'= 'next[j]	next[j]'= 'next[j]

Tab5 of Writers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(rlseAtPww)	next[j]'= 'next[j]

Tab6 of Writers: ∀j,

j= k	j≠ k
next[j]'=NextLabel(w2)	next[j]'= 'next[j]

Tab8 of Writers: ∀j,

j= k	j≠ k
next[j]'=END	next[j]'= 'next[j]

Tab9 of Writers: ∀j,

j= k	(j≠k) ∧ (j∈('w.set-w.set')) ∧		(j≠k) ∧ (j∉('w.set-w.set'))
	'next[j]=waitAtPww	'next[j]=waitAtPwr	
next[j]'=END	next[j]'=rlseAtPww	next[j]'=rlseAtPwr	next[j]'= 'next[j]

7.3.2 An Illustration of an Actual Table

As an example, we consider two readers and two writers at a time. The actual table for the Readers/Writers program will be described in a skeleton view as shown in figure 7-5, spreading all columns and rows of all of the four processes.

We leave some of the cells in the table blank for simplicity. The contents of the grids are the corresponding values shown in Figure 7-4. To fill the content of the table, we need to copy the representative columns three times for reader processes and two times for writer processes. We can see that the real table illustrated in figure 7-5 will be very large, but easy to complete.

Figure 7-5 An Illustration of a Real Table

	pID=1 ∧ IsReader ∧														pID=2 ∧ IsReader ∧	(pID=3 ∧ IsWriter) ∧					pID=4 ∧ IsWriter ∧
	(*next[1]=rl) ∧	*next[t]= waitAtPml	*next[1]= rlscAtPml	(*next[1]=r3) ∧	*next[1]= waitAtPwr	*next[1]= rlscAtPwr	(*next[1]=r4) ∧	*next[1] = r5	(*next[1]=r6) ∧	*next[1]= waitAtPm2	*next[1]= rlscAtPm2	*next[1] = r7	(*next[1] r8) ∧	(*next[1] r9) ∧	.....	(*next[5]=wl) ∧	*next[5]= waitAtPww	*next[5]= rlscAtPww	*next[5]=w2	(*next[5] =w3) ∧	.....
rdcnt' =																					
rd' =																					
wt' =																					
m.cnt'=																					
m.set'|																					
w.cnt' =																					
w.set'|																					
next'|																					

Chapter 8

Inspection of the Program

This chapter has three sections. The first section presents a proof that the program-function table meets the requirement of disjointness and completeness properties. The second section and the third section use the table to confirm the liveness property and the invariant property of the rewritten program respectively.

Note that the proof is done manually here, which is a very tedious job. However, this is intended as a first step towards automating the process.

8.1 The Completeness and Disjointness of the Table

As mentioned in Section 2.4, to correctly represent the program behavior, a program-function table must have the two properties: disjointness and completeness.

Definition [43]: Let C_1, ..., C_n be the condition predicates for each column in the normal header of the table, and let x_1, ..., x_m be the free variables that appear in the predicates, then

A table is complete if
$(\forall x_1, \ldots, x_m, (C_1 \vee C_2 \vee \ldots \vee C_n)) = \text{true}$

and disjoint if
$(\forall i, j, (i \in [1..n] \wedge j \in [1..n] \wedge i \neq j \Rightarrow (\forall x_1,\ldots,x_m,(C_i \wedge C_j))))) = \text{false}$

We start the inspection process by proving the following theorem.

Theorem 8.1:
The program-function table (figure 7-4) for the rewritten Readers/Writers program is complete and disjoint.

Note that we have organized the header as a tree where each node is associated with a partitioning of the space of the node above it. This way of construction guarantees correctness of this structure. We prove theorem 8.1 using the structure of the table header, from the top to the bottom and level by level. Figure 8-2 represents the normal header of the table with a multi-levels structure (taken from figure 7-4). Moreover, as explained in Section 7.1, each column condition of the table is the conjunction of the conditions in each level. For instance, the column conditions for the left four columns are:

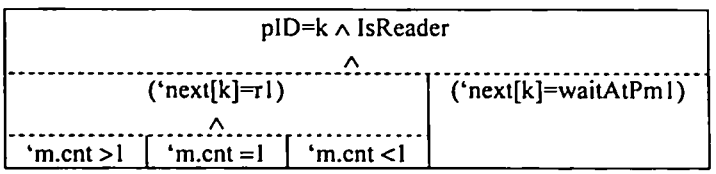


Figure 8-1 The Headers of the Left Four Columns

From left to right, the column conditions of the above four columns can be expressed as:

(pID=k ∧ IsReader) ∧('next[k]=r1) ∧ ('m.cnt >1)
(pID=k ∧ IsReader) ∧('next[k]=r1) ∧ ('m.cnt =1)
(pID=k ∧ IsReader) ∧('next[k]=r1) ∧ ('m.cnt <1)
(pID=k ∧ IsReader) ∧('next[k]=waitAtPm1)

Note that although we are looking at only one representative reader process and one representative writer process (see detailed explanation in Section 7.1), the following proof considers all processes in question.

Proof of the theorem:

For the 1^{st} level:

Only variable **pID** and expressions **IsReader** and **IsWriter** are used. The possible values of **pID** are integer numbers (see Section 6.6), which are the identifiers of the reader and writer processes. The two representative processes represent all of the processes in execution; that is, all of the possible values of **pID** appear exactly once and only once in the real table. Therefore, the 1^{st} level is complete since all possible values are covered. The table is also disjoint, because the **pID** of each process appears only once and the two Boolean expressions, **IsReader** and **IsWriter** tell the type of a process, which means that no two columns in the 1^{st} level can be true at the same time.

For the 2^{nd} level:

Only variable **next** with indices of process numbers appears. All possible values of array **next**: **r1, r2, ..., r9, w1, ..., w3,** including the special values of **waitAtPm1, rlseAtPm1, waitAtPm2, rlseAtPm2, waitAtPwr, rlseAtPwr, waitAtPww,** and **rlseAtPww**, appear exactly once and only once in the columns of each process. Therefore, the 2^{nd} level is complete. Here the disjointness property also holds because each of the possible values appears only once in the columns of each process, which means that no two columns of each process can be true at the same time.

For the 3^{rd} level:

We have different cases and we will go through all of the reader/writer columns as below:

Since $\mathbf{X_{11}}$: $(\forall x,(x>1 \vee x=1 \vee x<1))$=true,

$\mathbf{X_{21}}$: $(\forall x,(x>0 \vee x=0 \vee x<0))$=true,

$\mathbf{X_{31}}$: $(\forall x,(x\geq 1 \vee x<1))$=true;

$\mathbf{X_{12}}$: $(\forall x,((x>1 \wedge x=1) \vee (x=1 \wedge x<1) \vee (x>1 \wedge x<1))$=false,

$\mathbf{X_{22}}$: $(\forall x,((x>0 \wedge x=0) \vee (x=0 \wedge x<0) \vee (x>0 \wedge x<0))$=false,

$\mathbf{X_{32}}$: $(\forall x,(x\geq 1 \wedge x<1))$=false

Therefore,
Case 1: In columns where **next[i]=r1, next[i]=r4, next[i]=r6,** and **next[i]=r9**, only variable **m.cnt** appears. By substituting **x** with **m.cnt** in $\mathbf{X_{11}}$, $\mathbf{X_{12}}$ and $\mathbf{X_{21}}$, $\mathbf{X_{22}}$, respectively, we can prove the theorem; that is, the table at the 3rd level is complete and disjoint.

Case 2: In columns where **next[i]=r3** and **next[i]=r8**, variable **rdcnt** appears and there is a lower level involving variable **w.cnt**.
First, in the level with variable **rdcnt**, substituting **x** with **rdcnt** in $\mathbf{X_{11}}$, $\mathbf{X_{22}}$ and $\mathbf{X_{21}}$, $\mathbf{X_{22}}$, respectively, will prove the theorem; that is, the table is complete and disjoint at this level.

In the lower level with variable **w.cnt**, substituting **x** with **w.cnt** in $\mathbf{X_{11}}$, $\mathbf{X_{12}}$, $\mathbf{X_{21}}$, $\mathbf{X_{22}}$ and $\mathbf{X_{31}}$, $\mathbf{X_{32}}$, respectively, will prove the theorem; that is, the table is complete and disjoint at this level.

Case 3: In columns where **next[i]=w1** and **next[i]=w3**, only variable **w.cnt** appears. Substituting **x** with **w.cnt** in $\mathbf{X_{11}}$, $\mathbf{X_{12}}$ and $\mathbf{X_{21}}$, $\mathbf{X_{22}}$ will prove the theorem; that is, the table is complete and disjoint at this level.

Therefore, the 3rd level has been proved.

Thus, in summary, the table is complete and disjoint. Theorem 8.1 is proved.

Figure 8-2 The Normal Header of the Program-Function Table (in figure 7-4):

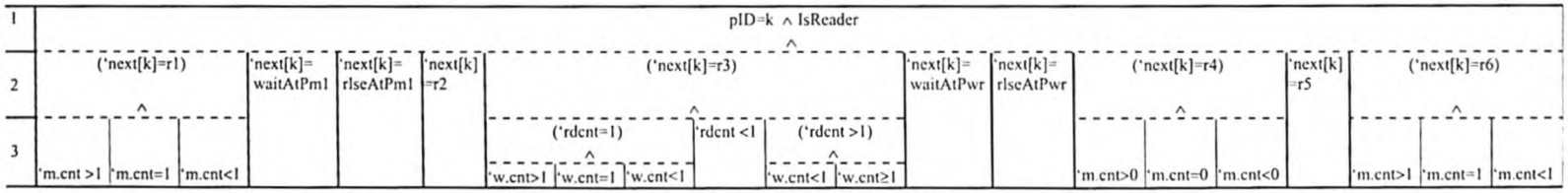
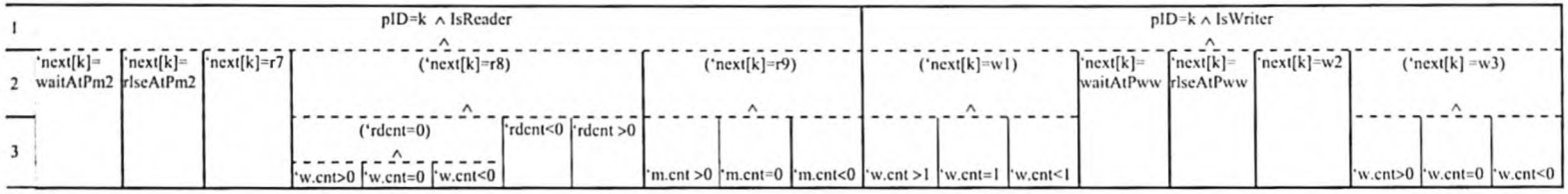


Figure 8-2 Continued



8.2 Showing Clean Completion

A concurrent program has a clean completing means that all of its constituent processes fully finish the execution.

8.2.1 Definition of Decreasing Quantity of Program State

In order to show that the program will not stop unless all processes have finished, we define a vector to describe the property of **Decreasing Quantity (DQ)** of the program states as following:

DQ = (Pros, IntRW(next)), which can also be expressed as

DQ = (Pros, IntRW(next[1]), IntRW(next[2]), ... IntRW(next[M]))

where
Pros represents the number of all reader and writer processes at the time;
M is the number of all processes;
IntRW() is the function that can map all possible values of the variable **next** to an integer, defined in figure 8-3.

DQ consists of two parts: the number of all existing processes and the mapping integers. Each integer is provided through the function of **IntRW()** for a given label value, which identifies the statement that is being executed in each process.

A concurrent program can be fully understood by looking at all of its constituent processes. A process in the concurrent program, once started, can be in three different states: suspended to wait for permission to enter a critical section, executing, or ready to advance. However, executing is not really going to be a state. We ignore the execution of a primitive statement and assume that it lasts zero time. Thus, there are really only two states, either ready to execute or waiting on a semaphore queue, for which the process has a label assigned to its current execution

state. For instance, the value of **next[i]** is the label for the **i**th process in its execution state.

Below is the definition of function **IntRW()**.

X	IntRW(x)
r1	15
waitAtPm1	14
rlseAtPm1	13
r2	12
r3	11
WaitAtPwr	10
RlseAtPwr	9
r4	8
r5	7
r6	6
waitAtPm2	5
rlseAtPm2	4
r7	3
r8	2
r9	1
w1	5
WaitAtPww	4
RlseAtPww	3
w2	2
w3	1
END[14]	0

Figure 8-3 The Definition of Function IntRW()

From the above definition, we have:

Corollary 1:

DQ = 0 if and only if

(Pros, IntRW(next[1]), IntRW(next[2]),... IntRW(next[M])) = 0 where

Pros = 0 means that there is no process at the state;

[14] "END" is introduced as a pseudo-label to indicate the end of execution of a process, which is not going to be a state. Detailed explanation can be found in Section 6.4.3.

IntRW(next[1])=0, IntRW(next[2])=0,..., and IntRW(next[M])=0 means that all processes arrive at the "END" label statement and finish all operations.

Figure 8-4 is a directed graph describing the order relation of all possible values of the variable **next**, which correspond to the labels of the statements. The order relation indicates possible sequences of the statement labels during the execution of the program, and is given only for the possible values of variable **next** for readers and writers. The branches occur because there are conditional statements in the program.

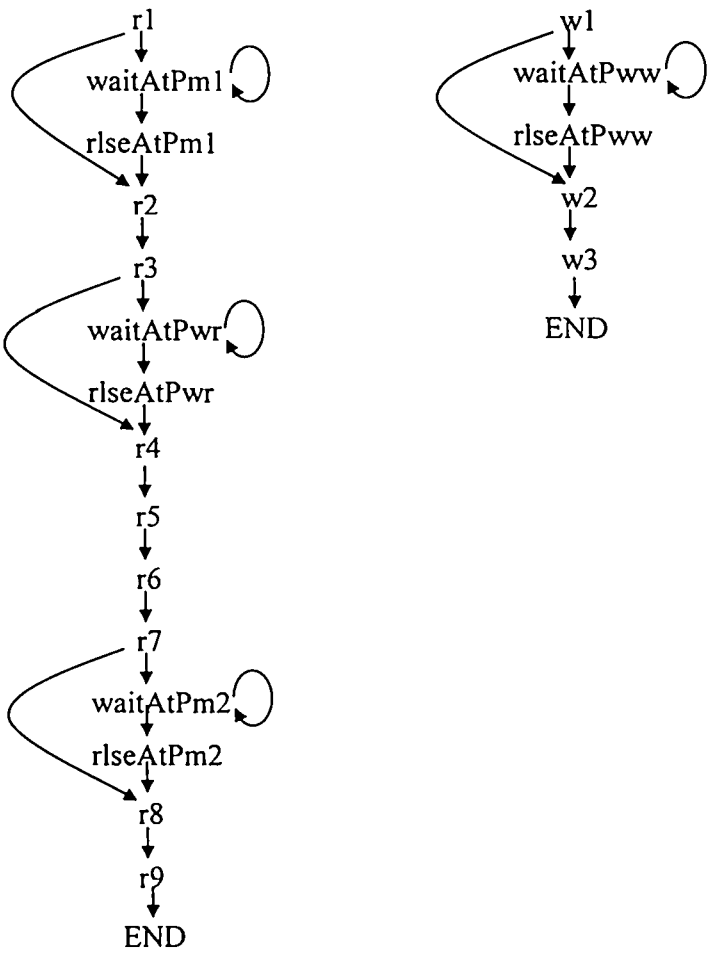


Figure 8-4 The Order Relation of All Possible Values of the Variable **next**

Note that, the readers/writers program is non-terminating and goes from start back to start. Each process of the program is assigned a set of labels during execution, from r1 to r9 for a reader or from w1 to w3 for a writer. After the last statement in a process with label r9 or w3 finishes, the execution of this process ends; a newly executing process, either a reader or a writer, will again start from r1 or w1.

From above figure, we have another corollary:

Corollary 2: Let x_1 and x_2 be two nodes in the above directed graph, and
we write "x_1 -) x_2", to stand for that "x_1 is above x_2",
If x_1 -) x_2, then $IntRW(x_1) > IntRW(x_2)$

The order property of DQ:

When the program is executed we will have a sequence of states. At each state we will have a set of processes. Supposing these processes are executed at some statements with corresponding label $Rx_1 \ldots Ri_1$, $Wx_1 \ldots Wi_1$, and $Rx_2 \ldots Ri_2$, $Wx_2 \ldots Wi_2$,
we write

$\Sigma r_1 = IntRW(Rx_1) + \ldots + IntRW(Ri_1)$
$\Sigma w_1 = IntRW(Wx_1) + \ldots + IntRW(Wi_1)$
$\Sigma r_2 = IntRW(Rx_2) + \ldots + IntRW(Ri_2)$
$\Sigma w_2 = IntRW(Wx_2) + \ldots + IntRW(Wi_2)$
$DQ_1 = (Pros_1, IntRW(Rx_1), \ldots, IntRW(Ri_1), IntRW(Wx_1), \ldots, IntRW(Wi_1))$
$DQ_2 = (Pros_2, IntRW(Rx_2), \ldots, IntRW(Ri_2), IntRW(Wx_2), \ldots, IntRW(Wi_2))$
$Pros_1$ is the number of all processes at state1;
$Pros_2$ is the number of all processes at state2.
DQdecrease stands for $(DQ_1 > DQ_2)$

Thus, the order property of **DQ** can be defined below:

	$Pros_1 > Pros_2$	$(Pros_1 = Pros_2) \wedge (\Sigma r_1 + \Sigma w_1) > (\Sigma r_2 + \Sigma w_2)$	$(Pros_1 = Pros_2) \wedge (\Sigma r_1 + \Sigma w_1) \leq (\Sigma r_2 + \Sigma w_2)$	$Pros_1 < Pros_2$
DQdecrease=	True	True	False	False

Figure 8-5 The Order Property of **DQ**

8.2.2 The Theorem of Decreasing Quantity

This section we present the theorem of Decreasing Quantity, and give the proof.

The theorem of DQ:

a) If there is a state change, DQ decreases;
b) If there is no state change possible, DQ is zero; and
c) If DQ is zero, there is no waiting process.

Assumption of the theorem:
There are no new readers/writers arriving.

Explanation of the terms:
"process state": A process can be described in terms of a set of state variables. At any given moment, each of the state variables will contain a particular value, and this collection of values is used to describe the state of the process.
"state change": This means that at least one variable of the process state has changed its value.
"waiting process": This process that cannot change the program state.

8.2.3 The Proof

Proof of the theorem of DQ:

For item **a)** of the theorem, we will give the proof column by column (see table 7-4) and in accordance with the values of array variable **next**.

(1) All of the gray columns, including columns 1, 7, 10, 12, 15, 19, 25, 28, and 30 of the reader processes and columns 1, 7 of writer processes, describe those cases that can never occur in the execution. Consequently, there is no state change. Therefore, the hypothesis is false and the theorem is true.

(2) In column 2 of readers, there is a state change; the value of variable **next** has changed.
We have:

($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))), and

'next[k]=r1, where **k** denotes the currently executing process.

If a process is currently active, the value of variable **next** is assigned to **NextLabel(r1)**, where **r1** denotes the current statement label of the process.

Then, from the definitions of function **NextLabel()** and function **IntRW()**, we have:

IntRW(r1) > IntRW(NextLabel(r1))

For all other processes that are waiting to advance their execution, there is no change of variable **next** and **IntRW(next)**, so the corresponding parts in the expression of the **DQ** definition for these processes keep the same values.

Because there are no new coming readers/writers, and no process finishes its execution, the number of existing processes is unchanged.

Columns **5, 6, 11, 14, 16, 18, 20, 23, 24, 26, and 29** of readers and columns **2, 4, 5, and 6** of writers are similar to column **2** of readers; thus, the proof remains the same.

Therefore, **'DQ > DQ'**, and **a)** is true in these cases.

(3) In column **3** of readers, there is a state change; the value of variable **next** has changed.
We have:

($\forall$j, ((j=k $\wedge$ next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$ next[j]'='next[j]))),

and, **'next[k]=r1** where **k** denotes the currently executing process,
and, when a process is currently active, the value of variable **next** is assigned to **waitAtPm1**.

From the definitions of function **NextLabel()** and function **IntRW()**, we have:

IntRW(r1) > IntRW(waitAtPm1)

For all other processes that are waiting to advance the execution, there is no change of the variable **next**, so the corresponding parts in the **DQ** definition for these processes keep the same values.

Because there are no new readers/writers arriving, and no process finishes its execution, the number of processes is unchanged.

Columns **9,** and **21** of readers and column **3** of writers are similar to column **3** of readers; thus, the proof remains the same.

Therefore, **'DQ > DQ'. a)** is true in these cases.

(4) Columns **4, 13,** and **22** of readers and column **4** of writers represent no state change in the program execution; therefore **a)** is true since the hypothesis is false.

(5) In column **17** of readers, there is a state change; the value of variable **next** has changed.
We have:

((∀j, (j=k ∧ next[j]'=NextLabel(r4)) ∨
(j≠k ∧ j∈('m.set-m.set') ∧'next[j]=waitAtPm1∧ next[j]'=rlseAtPm1)∨
(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))),
and **'next[k]=r4,** where **k** denotes the currently chosen process.

When a process is currently active, the value of variable **next** is assigned to **NextLabel(r4) r4,** where **r4** denotes the current statement label of the process.
When a process is activated from the semaphore's waiting list by a V-operation of another process, the value of variable **next** is assigned to **rlseAtPm1.**

From the definitions of function **NextLabel()** and **IntRW()**, we have:

IntRW(r4) > IntRW(NextLabel(r4))

IntRW(x) > IntRW(rlseAtPm1), where **x** is the before-value of **next** variable with its after-value of **rlseAtPm1**.

For all other processes that are waiting to advance their execution, there is no change of the variable **next**, so the corresponding parts in the expression of the **DQ** definition keep the same value.

Because there are no new coming readers/writers, and no process finishes its execution, the number of processes is unchanged.

Column **27** of readers is similar to column **17** of readers; thus, the proof remains the same.

Therefore **'DQ > DQ'. a)** is true in these cases.

(6) In column **31** of readers, there is a state change; the value of the variable **next** has changed.
We have:

(∀j, (j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])),

and **'next[k]=r9** where **k** denotes the currently executing process.

When a process is currently active, the value of the variable **next** is assigned to **END**, which means the process has no more state changes and the **DQ** of the chosen process is then equal to **zero**.

Then, from the definitions of function **NextLabel()** and function **IntRW()**, we have:

IntRW(r9) > IntRW(END)

All other processes keep the same value of the element in array **next.** When they finally advance to the same state as the currently chosen process, the **DQ** will get to **zero.**

There are no new coming readers/writers, and the current executing process finishes its execution, the number of process decreases by one.

Column **8** of writers is similar to column **31** of readers. Thus, the proof remains the same.
Therefore, **'DQ > DQ'. a)** is true in these cases.

(7) In column **32** of readers, there is a state change; the value of variable **next** has changed.
We have:

((∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('m.set-m.set') ∧'next[j]=waitAtPm2∧next[j]'=rlseAtPm2)
∨ (j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))),
and **'next[k]=r9**, where **k** denotes the currently executing process.

When a process is currently active, the value of the variable **next** is assigned to **END**, which means the process has no more state changes and the **DQ** of the chosen process is then equal to **zero**.

When a process is activated from the semaphore's waiting list by the V-operation of another process, the value of the variable **next** is assigned to **rlseAtPm2**.

Thus, from the definitions of function **NextLabel()** and function **IntRW()**, we have:

IntRW(r9) > IntRW(END)

IntRW(x) > IntRW(rlseAtPm2), where **x** is the before-value of the **next** variable and **rlseAtPm2** is the after-value.

All other processes keep the same value of the variable **next**, so the **DQ** remains the same value.

There are no new coming readers/writers, and the current executing process finishes its execution, the number of process decreases by one.

Column **9** of writers is similar to column **32** of readers. Thus, the proof remains the same.

Therefore, **'DQ > DQ'. a)** is true in these cases.

Till now, **a)** is proved for all columns.

For item b) of the theorem:

There are a few places where a specific process (the process with statement assigned label of waitAtPsem) cannot advance. They are columns **4, 13,** and **22** of readers and column **4** of writers.

(1) In each of these columns, the corresponding value of function IntRW(**next**) equals to one of the values 14,10,5, or 4 according to the definition (see figure 8-3), this follows that,

IntRW(**next**)>0;

We have DQ = (Pros, IntRW(**next**)), thus DQ is not zero.

(2) In each of these columns, although the current process is waiting on a semaphore, the non-positive value of the semaphore implies an execution of another process. This is because:

In column 4, and 22 of readers, we have

'next[k]=waitAtPm1, and 'next[k]=waitAtPm2 respectively, and the invariant

$(\exists i, i=k \wedge (next[i]=waitAtPm1 \vee next[i]=waitAtPm2)) \Rightarrow m.cnt<0;$

According to the definition of semaphore (see Section 6.4.3), m.cnt<0 implies there is at least one reader waiting for another reader's execution to invoke it from suspending. Thus, in these cases, there is possible state change.

In column 13 of readers and column 4 of writers, we have the invariant

$w.cnt<1 \Rightarrow ((wt=1 \wedge rd=0) \vee (rd \geq 1 \wedge wt=0) \vee (rd=0 \wedge wt=0 \wedge (\exists t, (next[t]=rlseAtPwr \vee next[t]=rlseAtPww))))$,

which indicates that, besides the waiting process, there is another process (at least) that is being executed: either a writer is writing, or a reader (or readers) is reading, or one process just finished its reading/writing that will invoke the waiting process. Thus, in these cases, there is possible state change.

Thus, from (1) and (2), we know that there is possible state change and DQ is not zero for these cases.

Therefore, **b)** is true.

For item c) of the theorem:
If one process is not at the end, the DQ will not be zero because the corresponding value of IntRW for this process will be a positive integer, i.e., one of the values in [15...1]. For a waiting process, the corresponding value of function IntRW(**next**) will equal to 14, or 10, 5, 4 (see the definition in figure 8-3). That is, in the definition

DQ = (Pros, IntRW(**next**)),

the part of function IntRW[**next(k)**] (where **k** is the identifier of currently executing process), is not zero. Therefore, if a process is not at the end, DQ is not zero.

Therefore, if DQ is zero, the values of IntRW(next) are zero, i.e., all processes come to the **END** label which follows that there is no waiting process.

Therefore, **c)** is true.

By now, the proof is complete.

8.3 Invariant Property Proof

Invariant properties are established by systematically checking that what are preserved by every primitive statement in the program. An invariant must be true

after initialization of the program variables and before, and after each execution of the primitive statements.

8.3.1 Description of Program Invariants

Using the "before/after" notation [42][52], we describe the theorem corresponding to program correctness:

If the requirement predicate ('wt=0 ∨ 'rd=0) ∧ ('wt<2) is true before the execution of each primitive statement, we must prove that the predicate (wt'=0 ∨ rd'=0) ∧ (wt'<2) will be true after the execution of each primitive statement.

Program correctness is not simply a matter of local behavior; rather, the whole program must be considered. The way we take global properties of the program into consideration is to include additional invariant properties needed to strengthen the invariant until it expresses all of the essential global properties. Strengthening the invariant strengthens not only the precondition for each primitive statement, but also the postcondition, which adds to the "proof obligations." [21]

We express the invariant as the conjunction of a set of clauses. The set of invariant clauses was not found at one time but through an iterative process. Having not captured all of the essential global properties, we repeatedly found that we needed to identify more invariant clauses each time to strengthen the invariant clause set, and this process continued until we could prove the theorem of invariance.

The clause set of the whole program invariant is described in figure 8-6.

Figure 8-6 The Program Invariant

	Invariant clause	The explanation	Initial value
rp_1	wt=0 ∨ rd=0	The requirement predicate of the program.	**True** with rd=0, wt=0
rp_2	wt <2	The requirement predicate of the program.	**True** with wt=0
V_1	rd ≥0	The number of active readers cannot be negative.	**True** with rd=0
V_2	wt ≥0	The number of active writers cannot be negative.	**True** with wt=0
V_3	rdcnt ≥0	The number of all reader processes, including the processes waiting to access to a critical section, cannot be negative.	**True** with rdcnt=0
V_4	w.cnt ≤1	The value of semaphore **w** cannot be greater than 1.	**True** with w.cnt=1
V_5	m.cnt ≤1	The value of semaphore **mutex** cannot be greater than 1.	**True** with m.cnt=1
V_6	w.cnt=1⇒ (wt=0 ∧ rd=0)	If the value of semaphore **w** is equal to 1, then there must be no active readers and writers.	**True** with w.cnt=1, wt=0, rd=0
V_7	rdcnt>1⇒ rd≥1	If more than one reader appear, then at least one reader is active in the w-critical-section (see Section 5.2).	**True** with rdcnt=0, rd=0
V_8	w.cnt<1⇒ ((wt=1∧rd=0) ∨ (rd≥1∧wt=0) ∨ (rd=0 ∧ wt=0∧(∃t, (next[t]=rlseAtPwr ∨ next[t]=rlseAtPww))))	If the value of semaphore **w** is less than 1, then there must be either readers or writers active in the w-critical-section, or that one process is activated from the waiting list.	**True** with w.cnt=1 wt=0, rd=0, (∀i, next[i])=r1
V_9	(∃i, i=k ∧ (next[i]=r3 ∨ next[i]=rlseAtPwr ∨ next[i]=r4∨ next[i]=r5 ∨ next[i]=r6∨ next[i]=rlseAtPm2∨ next[i]=r7)) ⇒ rdcnt>0	If any reader, when executed, has a label in [r3, rlseAtPwr, r4, r5, r6, rlseAtPm2, r7], then **rdcnt** is greater than 0.	**True** with (∀i, next[i])=r1, rdcnt=0
V_{10}	(∃i, i=k ∧ next[i]=rlseAtPwr) ⇒ (rd=0 ∧w.cnt<1)	If any reader, when executed, has a label of rlseAtPwr, then **rd** equals to 0 and the value of semaphore **w** is less than 1.	**True** with w. cnt=1, (∀i, next[i])=r1, rd=0

Figure 8-6 The Program Invariant (continued)

	Invariant clause	The explanation	Initial value
V_{11}	(∃i, i=k ∧(next[i]=rlseAtPm1 ∨ next[i]=r2 ∨ next[i]=r3 ∨ next[i]=rlseAtPwr ∨ next[i]=r4 ∨ next[i]=rlseAtPm2 ∨next[i]=r7 ∨ next[i]=r8 ∨ next[i]=r9)) ⇒ m.cnt<1	If any reader, when executed, has a label in [rlseAtPm1, r2, r3, rlseAtPwr, r4, rlseAtPm2, r7, r8, r9], then the value of semaphore **mutex** is less than 1.	**True** with (∀i, next[i])=r1, rdcnt=0
V_{12}	(∃i, i=k ∧(next[i]=r1 ∨next[i]=rlseAtPm1 ∨ next[i]=r2 ∨ next[i]=r4 ∨ next[i]=r5 ∨ next[i]=r6 ∨ next[i]=rlseAtPm2 ∨ next[i]=r7 ∨ next[i]=r9)) ⇒ rd=rdcnt	If any reader, when executed, has a label in [r1, rlseAtPm1, r2, r4, r5, r6, rlseAtPm2, r7, r9], then **rd** is equal to **rdcnt**.	**True** with (∀i,next[i])=r1, rdcnt=0, rd=0
V_{13}	(∃i, i=k ∧(next[i]=r3 ∨ next[i]=rlseAtPwr))⇒ rd=rdcnt-1	If any reader, when executed, has a label of r3 or rlseAtPwr, then **rd** is less than **rdcnt** by 1.	**True** with (∀i, next[i])=r1, rdcnt=0, rd=0
V_{14}	(∃i, i=k ∧next[i]=r8) ⇒ rd=rdcnt+1	If any reader, when executed, has a label of r8, then **rd** is greater than **rdcnt** by 1.	**True** with (∀i, next[i])=r1, rdcnt=0, rd=0
V_{15}	(∃i, next[i]=rlseAtPww) ⇒ (wt=0 ∧ w.cnt<1)	If any writer, when executed, has a label of rlseAtPwr, then **wt** is 0 and the value of semaphore **w** is less than 1.	**True** with (∀i, next[i]) =w1, wt=0, w.cnt=1
V_{16}	(∃i, (next[i]=w2 ∨ next[i]=w3)) ⇒ (wt=1 ∧ w.cnt<1)	If any writer, when executed, has a label of w2 or w3, then **wt** is equal to 1 and the value of semaphore **w** is less than 1.	**True** with (∀i, next[i]) =w1, wt=0, w.cnt=1
V_{17}	(∃i, i=k∧(next[i]=waitAtPm1∨next[i]=waitAtPm2))⇒m.cnt<0	If any reader, when executed, has a label of waitAtPm1 or waitAtPwr], then **m.cnt** is less than 0.	**True** with m.cnt=1; (∀i, next[i])=r1

8.3.2 Conditions before the Execution of Any Processes

Initially, all of the processes are at the starting state of their programs. All of the variables have been initialized properly, with the initial conditions indicating that there are no readers and no writers in action.

The initial values of the variables included in the invariant in Figure 8-6 are as follows:

rd=0;
wt=0;
rdcnt=0;
w.cnt=1; w.set=Φ;
m.cnt=1; m.set=Φ;
(∀i, next[i]) =r1 ∧ **IsReader** for reader process;
(∀i, next[i])=w1 ∧ **IsWriter** for writer process.

Note:
Symbol "Φ" is used to denote an empty set.
Different types of processes, such as readers and writers, will start the execution with different initial values of the variable **next**.

With the initial values, we can see that all of the invariant clauses are initially true by substituting the initial values in the expression. The rightmost column in figure 8-6 shows us the truth-value holds for each clause.

8.3.3 Notations Used in The Proof

We introduce the symbol " ⊢ "to represent the "derived by" relation in the proof:

Let P_1, P_2, ..., P_n (where n is a positive integer) be a set of predicate expressions, called premises, and Q be a predicate expression, called a conclusion. By applying

inference rules to the premises, we can draw a conclusion. This sequent is denoted by the following expression:

$$P_1, P_2, \ldots, P_3 \vdash Q$$

We introduce the notation of "≡" describe an equivalent replacement. For instance, when replacing the definition of the invariant "V_1", we use the "≡" symbol to indicate this action.

8.3.4 Proof of Sample Columns

This section will give the proof for sample columns. A complete proof is given in Appendix B. We will use the following expressions to make the proof more readable:

'INVARIANTS: describes all of the invariant clauses with before-notation for all of the variables in the expressions;

INVARIANTS': describes all of the invariant clauses with after-notation for all of variables in the expressions;

$\mathbf{V_i}$ (where **i=1...16,** see figure 8-6): describes a clause in the invariant set;

'V_i (where **i=1...16**, see figure 8-6): describes a clause in the invariant set with before-notation of all variables in the expressions;

V_i' (where **i=1...16**, see figure 8-6): describes a clause in the invariant set with after-notation of all variables in the expressions;

NC(Var_1,...Var_2): indicates that the variables in the list within the parentheses are not changed as the state changes, i.e. **Var_1'='Var_1, Var_2'='Var_2** and etc.

Columns of Reader Processes:

Column 1+32*(k-1) for k=1...n, where n is the number of readers [15]:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt>1) {column condition[16]}
⊢
'V_5
≡
'm.cnt≤1
⊢
¬('m.cnt>1)
⊢
¬((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt>1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 2+32*(k-1) for k=1...n, where n is the number of readers:

Prove: INVARIANTS'
Given: 'INVARIANTS
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1) {column condition}
∧ m.cnt'='m.cnt-1
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))
Proof:

---

[15] We use the clause to imply that we do not want a column to be "executed" twice in a row, because the same would remain true for the rest of the columns.

[16] The condition at the top of the column, which must be satisfied when executing the portion of the program corresponding to the column.

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change; hence, they will be true with the after-values of the variables.

For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

$'V_5$

≡

'm.cnt≤1

⊢ {substitute **(m.cnt'+1)** for **'m.cnt**}

m.cnt'+1≤ 1

⊢

m.cnt'≤ 0

⊢

m.cnt'≤1

≡

V_5'

$'V_8$
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace **$'V_8$** with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨ {bold font means replacement occuring
('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))) between current step and last step}
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0∧(∃t, (t=k∧('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))∨ (t≠k∧('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {substitute **corresponding after-values** for **'wt, 'rd, 'w.cnt, 'next** in **'V_8**}

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww))))

≡

V_8'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ (**'next[k]=r1**)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r1)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r1)

∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r1)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ 'next[k]=r1

∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)

∧ m.cnt'='m.cnt-1

⊢

'm.cnt= 1

∧ m.cnt'='m.cnt-1

$\vdash$

$m.cnt' = 0$

$\vdash$

$m.cnt' < 1$

$\vdash$

V_{11}'

$'V_{12}$

$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\equiv$ {replace 'V_{12} with its definition}

($\exists$i, i=k $\wedge$('next[i]=r1$\vee$'next[i]=rlseAtPm1$\vee$'next[i]=r2 $\vee$'next[i]=r4$\vee$'next[i]=r5

$\vee$'next[i]=r6$\vee$'next[i]=rlseAtPm2 $\vee$'next[i]=r7 $\vee$'next[i]=r9)) $\Rightarrow$ 'rd='rdcnt

$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(r1)**}

'rd='rdcnt

$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ 'next[k]=r1

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

rd'=rdcnt'

$\vdash$

V_{12}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=r3 $\vee$ next[i]'=rlseAtPwr))

$\vdash$

V_{13}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=r8)

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r1)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

{get function value of **NextLabel(r1)**}

{get function value of **NextLabel(r1)**}

{get function value of **NextLabel(r1)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

{get function value of **NextLabel(r1)**}

{get function value of **NextLabel(r1)**}

Chapter 9

Conclusions and Future Work

9.1 Summary

In this thesis, we presented a method that uses Tabular Expression to inspect concurrent programs. The inspection is based on (1) making the control pointers of the program explicit; (2) the preparation of a precise mathematical description, a program-function table, of the code; (3) the expression of a system invariant that captures the global properties in order to examine one process (column) at a time.

The approach is built mainly on the following facts:

1. A concurrent program, when executed, has equivalent non-deterministic sequential counterparts [5][6][13][14]. Every set of concurrent interacting processes (such as readers and writers) has the same behavioral effect as a single non-deterministic program.
2. John von Neumann's principle [39][60] states that the program state can be converted into data state.
3. The "Divide and Conquer" policy [48][53][57] helps eliminate the problem of overlooking possible cases when inspecting a complex product.
4. Tabular expressions [44][48][49][52] can be used to describe the program behavior in a more convenient and understandable fashion and make it easier to apply the "Divide and Conquer" principle.

9.2 Conclusions

The proposed approach has been successfully used to inspect the example program of the Readers/Writers problem. The main conclusions are as following:

1. A program-function table is able to describe the function of a concurrent program completely, precisely, and consistently, provided that the program behavior is fully captured in the variables in the table.

2. The program counter must be made explicit to help record extra information for analyzing the set of concurrent programs. Rewriting the program into a set of if-statements with the counter variable can move program state information into the data state, and can make the transfer of control from statement to statement explicit, which helps track the program execution.

3. The use of a system invariant is essential in capturing program structure and analyzing a set of concurrent processes. We found that a sufficiently strong invariant for the concurrent program is the key to a successful inspection.

4. A Decreasing Quantity (DQ) of program states can be used to prove the liveness property of the concurrent program, i.e., when there is a state change, DQ decreases; a zero value of DQ shows the clean completion of a program execution.

5. The techniques of invariant and Decreasing Quantity have been used for loops in sequential programs, but here we are using the same idea to do a very different job.

6. The key idea of our approach is the application of the "divide and conquer" policy through tabular expression. Based on this, we are able to inspect a concurrent program by going through each single column of the table respectively without referring to one another, and without going back and looking at the program.

7. Although this work involves mathematical treatments, it is essentially a disciplined and formal inspection approach rather than a mathematical proof. We were satisfied with assertions that were obvious from examining small sections of the program but did not insist on fully mechanical inference.

9.3 Future Work

In the future research, we can apply the method to examples in which the program contains potential defects that need to be detected.

The most challenging part of this method is to identify an invariant that is strong enough. In this work, we found that our process of recognizing the right invariant was somewhat iterative. Each unsuccessful proof attempt led to a closer investigation for a stronger invariant, which required necessary changes both on the table and on the invariant. However, this iterative process may be eliminated if the target program is fully understood when inspection starts.

The next step in this research would be to find automated assistance in the mechanical steps of verifying that the invariant does not change in each column. A research project[17] has been initiated where an interactive theorem-proving tool such as PVS has been used to automate the proof.

For the long run, we can try to find algorithms for extracting program invariant from the analysis of the program.

[17] David Parnas, Mark Lawford, and Vera Pantelic(graduate student), SQRL, Computing and Software Department, McMaster University

Bibliography

[1] "Table Tool System Developer's Guide." CRL Report 339&340, Software Engineering Research Group, McMaster University, Canada, 1997.

[2] R. Abraham. "Evaluating Generalized Tabular Expressions in Software Documentation" CRL Report 346, McMaster University, Canada, 1997.

[3] FA Ackerman, LS Buchwald, FH Lewski. "Software inspections: An effective verification process." IEEE Software Vol. 6, No. 3, 1989

[4] P. Anderson, T. Reps, T. Teitelbaum, M. Zarins. "Tool Support for Fine-Grained Software Inspection." IEEE Software, 2003.

[5] G. R. Andrews. "Concepts and Notations for Concurrent Programming." ACM Computing Surveys Vol. 15, No. 1, March 1983.

[6] G. R. Andrews. "Concurrent Programming: Principles and Practice." Benjamin - Cummings Publication, 1991.

[7] A. Aurum, H. Petersson, C. Wohlin. "State-of-the-art: Software Inspections after 25 Years." Software Testing Verification Reliability, Vol. 12, No. 3, September 2002.

[8] B. Bauer, D. L. Parnas. "Experience with the Use of Precise Documentation." IEEE, 1995.

[9] B.Berard, M.Bidoit, A.Finkel. *Systems and software verification model-checking techniques and tools.* Springer, 1999

[10] D. M. Berry. "Formal Specification and Verification of Concurrent Programs." Carnegie Mellon University, February 1993.

[11] L. Bic, A. C. Shaw. *The Logical Design of Operating Systems.* Prentice Hall, 1988.

[12] D. Bisant, J. Lyle. "A Two Person Inspection Method to Improve Programming Productivity." IEEE Transactions on Software Engineering Vol. 15, No. 10, 1989.

[13] A. Burns, G. Davies. *Concurrent Programming.* Addison-Wesley, 1993.

[14] D. Bustard, J. Elder, J. Welsh. *Concurrent Program Structures.* Prentice Hall, 1988.

[15] P. J. Courtois, F. Heymans, D. L. Parnas. "Concurrent Control with 'Readers' and 'Writers'." Communications of ACM, Vol. 14, No. 10, October 1971.

[16] N.G. De Bruijn, "Additional Comments on a Problem in Concurrent Programming, " Communications of ACM, Vol. 10, No. 3, March 1967.

[17] X. Deng, M. B. Dwyer. "Invariant-based Specification, Synthesis, and Verification of Synchronization in Concurrent Programs." http://www.cis.ksu.edu.

[18] E. W. Dijkstra. "Solution of a Problem in Concurrent Programming Control." Communications of ACM, Vol. 5, No. 9, September 1965.

[19] E. W. Dijkstra. "The Structure of the "THE" Multiprogramming System." Communications of the ACM, Vo. 11, No. 5, May 1968.

[20] E. W. Dijkstra. "Guarded Commands, Nondeterminacy, and Formal Derivation of Programs." Communications of the ACM, Vol. 18, No. 8, 1975.

[21] E. W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[22] A. Dunsmore, M. Roper, M. Wood. "Practical Code Inspection Techniques for Object-Oriented Systems: An Experimental Comparison." IEEE Software, 2003.

[23] M. Fagan. "Design and Code Inspections to Reduce Errors in Program Development." IBM Systems Journal, Vol. 15, No. 3, 1976.

[24] R. W. Floyd. "Assigning meanings to programs." Proceedings of Symposia in Applied Mathematics, 1967.

[25] B. T. Hailpern. *Verifying Concurrent Processes Using Temporal Logic.* Springer-Verlag, 1982.

[26] P. B. Hansen. "Concurrent Programming Concepts." Computing Surverys, Vol. 5, No. 4, December 1973.

[27] C.A.R. Hoare. "An Axiomatic Basis for Computer Programming." Communications of ACM, Vol. 12, No. 10, October 1969.

[28] R. Janicki. "On a Formal Semantics of Tabular Expressions." CRL report 355, McMaster University, Canada, October 1997.

[29] R. Janicki, R. Khedri. "On a Formal Semantics of Tabular Expressions." SERG 379, McMaster University, Canada, September 1999.

[30] R. Janicki, D. L. Parnas, J. Zucker. "Tabular Representations in Relational Documents." CRL report 313, McMaster University, Canada, December 1995.

[31] J. Knight, A. Myers. "An improved inspection technique." Communications of ACM, Vol. 11, No. 36, 1993.

[32] D. W. Knuth. "Additional Comments on a Problem in Concurrent Programming." Communications of ACM, Vol. 9, No. 5, May 1966.

[33] L. Lamport. "The 'Hoare Logic' of Concurrent Programs." Acta Informatica 14, 1980.

[34] L. Lamport. "Specifying Concurrent Program Modules." ACM Transactions of Programming Language and Systems, Vol. 5, No. 2, 1983.

[35] L. Lamport. "Win and Sin: Predicate Transformers for Concurrency." Technical Report 17, Digital Systems Research Center, Palo Alto, CA, 1987.

[36] F. Macdonald, J. Miller. "A Software Inspection Process Definition Language and Prototype Support Tool." Software Testing, Verification and Reliability, Vol. V, 1997.

[37] J. Martin, W. T. Tsai. "N-fold inspection: A requirements analysis technique." Communications of ACM, Vol. 33, No. 2, 1990.

[38] H. D. Mills. "The New Math of Computer Programming." Communications of the ACM, vol. 18, No. 1, January 1975.

[39] H. D. Mills. "Function Semantics for Sequential Programs." IFIP Congress 1980.

[40] S. Owicki. *Axiomatic Proof Techniques for Parallel Programs.* Garland Pub. 1980.

[41] S. Owicki, D. Gries. "Verifying Properties of Parallel Programs-An Axiomatic Approach." Communications of ACM, Vol. 19, No. 6, May 1976.

[42] D. L. Parnas. "Some Conclusions from an Experiment in Software Engineering Techniques." Proceedings of the 1972 FJCC.

[43] D. L. Parnas. "Some Theorems We Should Prove." CRL Report, McMaster University, Canada, 1992.

[44] D. L. Parnas. "Tabular Representation of Relations." CRL report 260, McMaster University, Canada, October 1992.

[45] D. L. Parnas. "Mathematics of Computation for software and other Engineers." CRL report, McMaster University, Canada, June 1993.

[46] D. L. Parnas. "Predicate Logic for Software Engineering." IEEE Transactions on Software Engineering, Vol. 19, No. 9, September 1993.

[47] D. L. Parnas. "Mathematical Description and Specification of Software." Proceedings of IFIP World Congress, 1994.

[48] D. L. Parnas. "Inspection of Safety-Critical Software Using Program-Function Tables." CRL report, McMaster University, Canada, June 1995.

[49] D. L. Parnas. *Software Fundamentals.* Ed. by Hoffman, D. M., Weiss, D. M. Addison-Wesley, March 2001.

[50] D. L. Parnas, J. Madey. "Functional Documentation for Computer Systems Engineering (version 2)." Science of Computer Programming, September 1993.

[51] D. L. Parnas, J. Madey, M. Iglewski. "Formal Documentation of Well-Structured Programs." CRL Report 259, McMaster University, Canada, September 1992.

[52] D. L. Parnas, J. Madey, M. Iglewski. "Precise Documentation of Well-Structured Programs." IEEE Transactions on Software Engineering, Vol. 20, No. 12, December 1994.

[53] D. L. Parnas, M. Lawford. "Inspection's role in software Quality Assurance." IEEE Software, 2003.

[54] D. L. Parnas, A. Lawton. "Precisely Annotated Hierarchical Pictures of Programs." CRL Report 359, McMaster University, Canada, 1998.

[55] D. L. Parnas, J. E. Shore, D. M. Weiss. "Abstract Types Defined as Classes of Variables." Proc. Conf. On Data: Abstraction, Definitin and Structure, Salt Lake City, Utah, 1976

[56] D. L. Parnas, A. J. Van Shouwen, S. P. Kwan. "Evaluation of Safety-Critical Software." Communications of the ACM, Vol. 33, No. 6, June 1990.

[57] D. L. Parnas, D. M. Weiss. "Active design reviews: Principles and practices." IEEE Computer Society, Los Alamitos, 1985.

[58] A. Porter, H. Siy, L. Votta. "A Review of Software Inspections." University of Maryland, and AT&T Bell Laboratories, October 1995.

[59] A. J. Van Schouwen. "The A-7 Requirements Model: Re-Examination for Real-Time Systems and an Application To Monitoring Systems." CRL Report 242, McMaster University, Canada, May 1992.

[60] J. Von Neumann. *Papers of John Von Neumann on Computing and Computer Theory.* Vol. 12 in the Charles Babbage Institute Reprint Series for the History of Computing, edited by Aspray, W. and Burks, A., 1986.

Appendix A

Tabular Representation of the SET Specification

Type[54] SET

Operations:

ISEMPTY(s, result): to check whether s is empty.

	$\|`s\|^{18}=0$	$\|`s\|\neq 0$
s'=	`s	`s
result'=	true	false

ADD(s, x, added): to add an element x to the set s.

	$`x \notin `s$	$`x \in `s$
s'=	$`s \cup \{x\}$	`s
added'=	true	false

$\wedge$ NC(x)

CHOOSE(s, x, chosen): to randomly choose one element from s, return that element x of s and remove x from s.

	$\|`s\| \neq 0$	$\|`s\| = 0$
s'=	`s-{x}	$\varnothing$
x'|	$(x' \in `s) \wedge (x' \notin s')$	$(x' = `x)$
chosen' =	true	false

[18] "|s|" denotes the cardinality of set s.

Appendix B

The Complete Proof of the Invariant Property of the Program

<u>Column 3+32*(k-1) for k=1...n, where n is the number of Readers:</u>
Prove: INVARIANTS'
Given 'INVARIANTS
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1) *{column condition}*
∧ m.cnt'='m.cnt-1
∧ m.set'='m.set ∪ {k}
∧ (∀j, (j=k ∧ next[j]'=waitAtPm1) ∨ (j≠k ∧ next[j]'='next[j]))
proof:
Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables. For the invariant clauses that have variables changed in the state change, we will prove them one at a time as below:
'V_5
≡
'm.cnt≤1
⊢ *{substitute 'm.cnt for (m.cnt'+1)}*
m.cnt'+1≤ 1
⊢
m.cnt'≤ 0
⊢
m.cnt'≤1
≡
V_5'

'V_8
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
≡ *{replace 'V_8 with its definition}*
('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
{bold font means replacement between current step and last step}
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t=k$\vee$ t$\neq$k)$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))

$\wedge$ ('next[k]=r1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t=k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))

$\vee$ (t$\neq$k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))

$\wedge$ ('next[k]=r1) $\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0$\wedge$ ($\exists$t, False$\vee$ (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))

$\wedge$ ('next[k]=r1) $\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))

$\wedge$ ('next[k]=r1) $\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ *{substitute corresponding after-values for 'wt, 'rd, 'w.cnt, 'next}*

w.cnt'<1$\Rightarrow$ ((wt'=1$\wedge$rd'=0) $\vee$ (rd'$\geq$1$\wedge$wt'=0)$\vee$(rd'=0 $\wedge$wt'=0 $\wedge$($\exists$t, (t$\neq$k$\wedge$ (next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww)))))

$\vdash$

V_8'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt<1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('next[k]=r1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=r3$\vee$next[i]'=rlseAtPwr$\vee$next[i]'=r4$\vee$next[i]'=r5$\vee$next[i]'=r6$\vee$next[i]'=rlseAtPm2$\vee$next[i]'=r7))

$\vdash$

V_9'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r1 $\wedge$ 'm.cnt<1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('next[k]=r1)

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm1) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg(\exists i, i=k \wedge next[i]'=rlseAtPwr)$

$\vdash$

V_{10}'

$((pID=k \wedge IsReader) \wedge `next[k]=r1 \wedge `m.cnt<1)$
$\wedge m.cnt'=`m.cnt-1$

$\vdash$

$`m.cnt< 1 \wedge m.cnt'=`m.cnt-1$

$\vdash$

$m.cnt'< 0$

$\vdash$

$m.cnt'< 1$

$\vdash$

V_{11}'

$`V_{12}$
$\wedge NC(rdcnt, rd, wt, w.cnt, w.set)$
$\wedge ((pID=k \wedge IsReader) \wedge `next[k]=r1 \wedge `m.cnt<1)$
$\wedge (\forall j, ((j=k \wedge next[j]'= waitAtPm1) \vee (j\neq k \wedge next[j]'=`next[j])))$

$\equiv$ *{replace V_{12} with its definition}*

$(\exists i, i=k \wedge (`next[i]=r1 \vee `next[i]=rlseAtPm1 \vee `next[i]=r2 \vee `next[i]=r4 \vee `next[i]=r5 \vee `next[i]=r6 \vee `next[i]=rlseAtPm2 \vee `next[i]=r7 \vee `next[i]=r9)) \Rightarrow `rd=`rdcnt$
$\wedge NC(rdcnt, rd, wt, w.cnt, w.set)$
$\wedge ((pID=k \wedge IsReader) \wedge `next[k]=r1 \wedge `m.cnt<1) \wedge (\forall j, ((j=k \wedge next[j]'= waitAtPm1) \vee (j\neq k \wedge next[j]'=`next[j])))$

$\vdash$

$`rd=`rdcnt \wedge NC(rdcnt, rd, wt, w.cnt, w.set)$

$\vdash$

$rd'=rdcnt'$

$\vdash$

V_{12}'

$NC(rdcnt, rd, wt, w.cnt, w.set)$
$\wedge ((pID=k \wedge IsReader) \wedge `next[k]=r1 \wedge `m.cnt<1)$
$\wedge (\forall j, ((j=k \wedge next[j]'= waitAtPm1) \vee (j\neq k \wedge next[j]'=`next[j])))$

$\vdash$

$NC(rdcnt, rd, wt, w.cnt, w.set) \wedge (\forall j, ((j=k \wedge next[j]'= waitAtPm1) \vee (j\neq k \wedge next[j]'=`next[j])))$
$\wedge \neg(\exists i, i=k \wedge (next[i]'=r3 \vee next[i]'=rlseAtPwr))$

$\vdash$

V_{13}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧next[i]'=r8)
⊢
V_{14}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧next[i]'=rlseAtPww)
⊢
V_{15}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))
⊢
V_{16}'

((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt<1)
∧ m.cnt'='m.cnt-1
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
'm.cnt< 1 ∧ m.cnt'='m.cnt-1
∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
m.cnt'< 0 ∧ (∀j, ((j=k ∧next[j]'= waitAtPm1) ∨ (j≠k ∧next[j]'='next[j])))
⊢
V_{17}'

Column 4+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=waitAtPm1) {column condition}

∧ (∀j, ((j=k ∧ next[j]'='next[j]) ∨ (j≠k ∧ next[j]'='next[j])))

proof:

Since all of the variables in all of the invariant clauses are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

Column 5+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1) {column condition}

∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_5, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_8

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace for 'V_8 with its definition}

('w.cnt<1⇒(('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0∧'wt=0∧(∃t, ('next[t]=rlseAtPwr∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtpm1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒(('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ('next[k]=rlseAtPm1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0∧'wt=0∧(∃t, (t=k∨ t≠k)∧('next[t]=rlseAtPwr∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPm1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0∧(∃t, (t=k∧('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))∨(t≠k ∧('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPm1)∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPm1)∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ *{substitute corresponding after-values for 'wt, 'rd, 'w.cnt, 'next in 'V_8}*

w.cnt'<1⇒((wt'=1∧rd'=0)∨(rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr∨next[t]'=rlseAtPww)))))

⊢

w.cnt'<1⇒((wt'=1∧rd'=0)∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0∧(∃t, (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww))))

≡

V_8'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ *{get function value of **NextLabel(rlseAtPm1)**}*

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ('next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'
NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm1
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

$'V_{11}$
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace $'V_{11}$ with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9)) ⇒'m.cnt<1
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm1
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(rlseAtPm1)**}

'm.cnt<1
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

m.cnt'<1∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{11}'

$'V_{12}$
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace for V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1 ∨'next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9)) ⇒ 'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm1
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(rlseAtPm1)**}

'rd='rdcnt ∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, (j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

rd'=rdcnt'

⊢

V_{12}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)

$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(rlseAtPm1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge (\forall j, ((j=k \wedge next[j]'=r2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge(next[i]'=r3 \vee next[i]'=rlseAtPwr))$

$\vdash$

V_{13}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=rlseAtPm1)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(rlseAtPm1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge (\forall j, ((j=k \wedge next[j]'=r2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge next[i]'=r8)$

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=rlseAtPm1)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(rlseAtPm1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge (\forall j, ((j=k \wedge next[j]'=r2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge next[i]'=rlseAtPww)$

$\vdash$

V_{15}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=rlseAtPm1)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(rlseAtPm1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge (\forall j, ((j=k \wedge next[j]'=r2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge(next[i]'=w2 \vee next[i]'=w3))$

$\vdash$

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(rlseAtPm1)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 6+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2) {column condition}
∧ rdcnt'='rdcnt+1
∧ (∀j, (j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_4, V_5, V_6 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_3
∧ rdcnt'='rdcnt+1

≡

'rdcnt≥0 ∧ rdcnt'='rdcnt+1

⊢ {substitute for **'rdcnt** with **rdcnt'-1**}

rdcnt' ≥ 1

⊢

rdcnt' ≥ 0

≡

V_3'

'V_7∧'V_{12}
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ rdcnt'='rdcnt+1

≡ {replace 'V_7, 'V_{12} with its definition}

'rdcnt>1⇒ 'rd≥1

∧(∃i, i=k ∧(‘next[i]=r1∨‘next[i]=rlseAtPm∨‘next[i]=r2∨‘next[i]=r4∨‘next[i]=r5∨‘next[i]=r6∨‘next[i]=rlseAtPm2∨‘next[i]=r7∨‘next[i]=r9))⇒‘rd=‘rdcnt
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r2)
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ rdcnt’=‘rdcnt+1

⊢

‘rdcnt>1⇒ ‘rd≥1
∧ ‘rd=‘rdcnt
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ rdcnt’=‘rdcnt+1

⊢ (substitute for ‘rdcnt, ‘rd with rdcnt’-1, rd’}

(rdcnt’-1 >1 ⇒ rd’≥1)
∧ (rd’=rdcnt’-1)

⊢

(rdcnt’>2 ⇒ rd’≥1)
∧ (rdcnt’=2 ⇒ rd’= 1)

⊢

(rdcnt’≥2 ⇒ rd’≥1)

⊢

(rdcnt’>1 ⇒ rd’≥1)

≡

V_7’

‘V_8
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r2)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(r2)) ∨ (j≠k ∧next[j]’=‘next[j])))

≡ {replace ‘V_8 with its definition}

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0) ∨ (‘rd≥1∧‘wt=0)∨ (‘rd=0 ∧‘wt=0 ∧(∃t, (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww)))))
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r2) ∧ (∀j, ((j=k ∧next[j]’=NextLabel(r2)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0) ∨ (‘rd≥1∧‘wt=0)∨(‘rd=0 ∧‘wt=0 ∧ (∃t, (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww)))))
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (‘next[k]=r2)∧ (∀j, ((j=k ∧next[j]’=NextLabel(r2)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨(‘rd=0 ∧‘wt=0 ∧ (∃t, (t=k∨ t≠k)∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww)))))
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ (‘next[k]=r2) ∧ (∀j, ((j=k ∧next[j]’=NextLabel(r2)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨
(‘rd=0 ∧‘wt=0 ∧ (∃t, (t=k ∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))∨ (t≠k ∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))

$\wedge$ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r2) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$
('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, False $\vee$ (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))

$\wedge$ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r2) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))

$\wedge$ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r2) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ *{substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values}*

w.cnt'<1$\Rightarrow$ ((wt'=1$\wedge$rd'=0) $\vee$ (rd'$\geq$1$\wedge$wt'=0)$\vee$(rd'=0 $\wedge$wt'=0 $\wedge$($\exists$t, (t$\neq$k$\wedge$ (next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww)))))

$\vdash$

w.cnt'<1$\Rightarrow$ ((wt'=1$\wedge$rd'=0) $\vee$ (rd'$\geq$1$\wedge$wt'=0) $\vee$ (rd'=0 $\wedge$wt'=0 $\wedge$($\exists$t, (next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww))))

$\equiv$

V_8'

'V_3
$\wedge$ rdcnt'='rdcnt+1

$\equiv$ {replace 'V_3 with its definition}

'rdcnt $\geq$ 0 $\wedge$ rdcnt'='rdcnt+1

$\vdash$

rdcnt'$\geq$ 1

$\vdash$

rdcnt'> 0

$\vdash$

V_9'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(r2)**}

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r2
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r3) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPwr)

$\vdash$

V_{10}'

$`V_{11}$
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace $`V_{11}$ with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9)) ⇒'m.cnt<1
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r2
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

'm.cnt<1
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ 'next[k]=r2
∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

m.cnt'<1∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{11}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r1 ∨next[i]'=rlseAtPm1∨ next[i]'=r2 ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨ next[i]'=r9))

⊢

V_{12}'

$`V_{12}$
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ rdcnt'='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace $`V_{11}$ with its definition}

(∃i, i=k ∧('next[i]=r1∨'next[i]=rlseAtPm1∨'next[i]=r2∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9))⇒'rd='rdcnt
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ 'next[k]=r2
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

'rd='rdcnt∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {substitute for 'rd, 'rdcntt with **corresponding after-values**}

rd'=rdcnt'-1∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{13}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]=r8))

⊢

V_{14}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r2)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r2)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of NextLabel(r2))
NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r3) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))
⊢
V_{17}'

Column 7+32*(k-1) for k=1...n, where n is the number of Readers:

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt>1) {column condition}
⊢
'V_4
≡
'w.cnt≤1
⊢
¬('w.cnt>1)
⊢
¬ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt>1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 8+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1) {column condition}
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ rd'='rd+1
∧ w.cnt'='w.cnt-1
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_6
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\equiv$

('w.cnt=1$\Rightarrow$ ('wt=0 $\wedge$ 'rd=0))
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt=1 $\wedge$ 'w.cnt=1)$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

'wt= 0 $\wedge$ 'rd=0

$\vdash$ {substitute wt' for 'wt}

wt'= 0

$\vdash$

wt'= 0 $\vee$ rd'=0

$\equiv$

rp_1'

'V_1
$\wedge$ rd'='rd+1

$\equiv$

'rd$\geq$0 $\wedge$ rd'='rd+1

$\vdash$ {substitute rd'-1 for 'rd}

rd'-1$\geq$0

$\vdash$

rd'$\geq$1

$\vdash$

rd'$\geq$0

$\equiv$

V_1'

'V_4
$\wedge$ w.cnt'='w.cnt-1

$\equiv$

'w.cnt$\leq$ 1 $\wedge$ w.cnt'='w.cnt-1

$\vdash$ (substitute w.cnt'+1 for 'w.cnt)

w.cnt'+1$\leq$1

$\vdash$

w.cnt'$\leq$ 0

$\vdash$

w.cnt'$\leq$ 1

$\equiv$

V_4'

$`V_6$
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ w.cnt'='w.cnt-1
⊢
'w.cnt=1 ∧ w.cnt'='w.cnt-1
⊢ {substitute **w.cnt'+1** for **'w.cnt**}
w.cnt'=0
⊢
¬(w.cnt'= 1)
⊢
V_6'

$`V_7$
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
⊢
'rdcnt=1
⊢ {substitute **rdcnt'** for **'rdcnt**}
rdcnt'=1
⊢
¬(rdcnt'> 1)
⊢
V_7'

$`V_6$
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ w.cnt'='w.cnt-1
∧ rd'='rd+1
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))
≡ {replace 'V_6 with its definition}
'w.cnt=1 ⇒ ('wt=0 ∧ 'rd=0)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ w.cnt'='w.cnt-1 ∧ rd'='rd+1 ∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))
⊢
('wt=0 ∧ 'rd=0)
∧ 'w.cnt=1
∧ NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ w.cnt'='w.cnt-1 ∧ rd'='rd+1 ∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))
⊢ {substitute for **'rd, 'w.cnt, 'wt** with **corresponding after-values**}

w.cnt'=0 ∧ (rd'=1 ∧ wt'=0)

⊢

w.cnt'<1 ∧ (rd'≥1 ∧ wt'=0)

⊢

V_8'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)

⊢

rdcnt'= 1

⊢

rdcnt'> 0

⊢

V_9'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r3)**}

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

'V_{11}
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9)) ⇒'m.cnt<1
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r3)**}

'm.cnt<1

∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

m.cnt'<1∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{11}'

'V_{13}
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ rd'='rd+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_{13} with its definition}

(∃i, i=k ∧('next[i]=r3∨'next[i]=rlseAtPwr)) ⇒'rd='rdcnt-1
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r3)**}

'rd='rdcnt-1
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {substitute for **'rd, 'rdcnt** with **corresponding after-values**}

rd'=rdcnt'∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{12}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r3)**}

NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r4 ∨next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

{get function value of **NextLabel(r3)**}

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

{get function value of **NextLabel(r3)**}

⊢

NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

{get function value of **NextLabel(r3)**}

⊢

NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt=1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

{get function value of **NextLabel(r3)**}

⊢

NC(rdcnt, wt, m.cnt, m.set, w.set) ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 9+32*(k-1) for k=1...n, where n is the number of Readers:</u>
Prove: INVARIANTS'

Given

'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1) {column condition}

∧ NC(rdcnt, rd, wt, m.cnt, m.set)

∧ w.cnt'='w.cnt-1

∧ w.set'='w.set ∪ {k}

∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_4

∧ w.cnt'='w.cnt-1

≡

'w.cnt≤ 1 ∧ w.cnt'='w.cnt-1

⊢ {substitute **w.cnt'+1** for **'w.cnt**}

w.cnt'+1≤1

⊢

w.cnt'≤ 0

⊢

w.cnt'≤ 1

≡

V_4'

'V_4

≡

'w.cnt≤ 1 ∧ w.cnt'='w.cnt-1

⊢ {substitute **w.cnt'+1** for **'w.cnt**}

w.cnt'+1≤1

⊢

w.cnt'≤ 0

⊢

¬(w.cnt'=1)

⊢

V_6'

'V_8

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)

∧ NC(rdcnt, rd, wt, m.cnt, m.set)

∧ w.cnt'='w.cnt-1
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

≡ {replace 'V_8 with its definition}

'w.cnt<1 ⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t,('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))))
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ NC(rdcnt, rd, wt, m.cnt, m.set) ∧ w.cnt'='w.cnt-1∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢

'w.cnt<1
∧(('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t,('next[t]=rlseAtPwr∨'next[t]=rlseAtPww))))
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ NC(rdcnt, rd, wt, m.cnt, m.set) ∧ w.cnt'='w.cnt-1∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢ {substitute for 'w.cnt, 'wt, 'rd, 'next with corresponding after-values}

w.cnt'<0
∧ ((wt'=1∧rd'=0)∨(rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t,(next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww))))

⊢

V_8'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)

⊢

rdcnt'= 1

⊢

rdcnt'> 0

⊢

V_9'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

$`V_{11}$
∧ NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9))⇒'m.cnt<1
∧ NC(rdcnt, rd, wt, m.cnt, m.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))

⊢

'm.cnt<1 ∧ NC(rdcnt, rd, wt, m.cnt, m.set)

⊢

m.cnt'<1

⊢

$`V_{11}$'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=r1 ∨next[i]'=rlseAtPm1∨ next[i]'=r2 ∨ next[i]'=r4 ∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨ next[i]'=r9))

⊢

$`V_{12}$'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

$`V_{13}$'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j]))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt=1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set)∧ (∀j, ((j=k ∧next[j]'=waitAtPwr) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

Column 10+32*(k-1) for k=1...n, where n is the number of Readers:

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt<1) {column condition}

⊢

'V_3 ∧ 'V_9
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt<1)

⊢ {replace 'V_3 ,'V_9 with its definition}

('rdcnt≥0)
∧ (∃i, i=k ∧('next[i]=r3 ∨'next[i]=rlseAtPwr∨'next[i]=r4 ∨'next[i]=r5 ∨'next[i]=r6 ∨'next[i]=rlseAtPm2∨'next[i]=r7))⇒'rdcnt>0
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt<1)

⊢

'rdcnt>0

⊢

¬('rdcnt<1)

⊢

¬ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt<1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 11+32*(k-1) for k=1…n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1) {column condition}
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ rd'='rd+1
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_4, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧'V_8 ∧ 'V_7
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

≡ {replace 'V_8, 'V_7 with its definition}

((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ ('rdcnt>1⇒ 'rd≥1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

('next[k]=r3 ∧'rdcnt>1 ∧ 'w.cnt<1)
∧ ('w.cnt<1⇒ ((wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ ('rdcnt>1⇒ 'rd≥1)∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

('next[k]=r3 ∧'rdcnt>1 ∧ 'w.cnt<1)
∧ ('w.cnt<1⇒ ((wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ 'rd≥1

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

$\vdash$

'rd$\geq$1 $\wedge$'wt=0 $\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

$\vdash$

'wt=0 $\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

$\vdash$

wt'= 0

$\vdash$

wt'= 0 $\vee$ rd'=0

$\equiv$

rp_1'

'V_1
$\wedge$ rd'='rd+1

$\vdash$

'rd$\geq$0 $\wedge$ rd'='rd+1

$\vdash$

rd'-1$\geq$0

$\vdash$

rd'$\geq$1

$\vdash$

rd'$\geq$0

$\equiv$

V_1'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ w.cnt'='w.cnt

$\vdash$

'w.cnt<1

$\vdash$

w.cnt'<1

$\vdash$

$\neg$(w.cnt'= 1)

$\vdash$

V_6'

'V_7
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

{substitute **wt'** for **'wt**}

{substitute **rd'-1** for **'rd**}

{substitute **w.cnt'** for **'w.cnt**}

$\wedge$ rd'='rd+1

$\equiv$ {replace 'V_7 with its definition}

('rdcnt>1$\Rightarrow$ 'rd$\geq$1)
$\wedge$ 'rdcnt>1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ rd'='rd+1

$\vdash$

'rdcnt>1 $\wedge$ 'rd$\geq$1

$\vdash$ {substitute corresponding after-values for 'rdcnt, 'rd }

rdcnt'>1 $\wedge$ rd'-1$\geq$1

$\vdash$

rdcnt'>1 $\wedge$ rd'$\geq$2

$\vdash$

rdcnt'>1 $\wedge$ rd'$\geq$1

$\vdash$

V_7'

'V_8 $\wedge$ 'V_7
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ rd'='rd+1
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))

$\equiv$ {replace 'V_7, 'V_8 with its definition}

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ ('rdcnt>1$\Rightarrow$ 'rd$\geq$1)
$\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ rd'='rd+1
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))

$\vdash$

'w.cnt<1$\wedge$ ('rd$\geq$1$\wedge$'wt=0)
$\wedge$ 'next[k]=r3 $\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ rd'='rd+1$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))

$\vdash$ {substitute for 'wt, 'rd, 'w.cnt with corresponding after-values}

w.cnt'<1 $\wedge$ (rd'-1$\geq$1 $\wedge$ wt'=0)

$\vdash$

w.cnt'<1 $\wedge$ (rd'$\geq$1 $\wedge$ wt'=0)

$\vdash$

w.cnt'<1 $\wedge$ ((wt'=1$\wedge$rd'=0)$\vee$(rd'$\geq$1$\wedge$wt'=0) $\vee$('rd=0 $\wedge$wt'=0 $\wedge$(next[a]'=rlseAtPwr $\vee$'next[a]=rlseAtPww)))

$\vdash$

V_8'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)

⊢

rdcnt'>1

⊢

rdcnt'> 0

⊢

V_9'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))

⊢ {get function value of NextLabel(r3)}

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

'V_{11}
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j]))

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9))⇒'m.cnt<1
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

'm.cnt<1 ∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

m.cnt'<1

⊢

V_{11}'

'V_{13}
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\wedge$ rd'='rd+1
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))

$\equiv$

($\exists$i, i=k $\wedge$('next[i]=r3$\vee$'next[i]=rlseAtPwr)) $\Rightarrow$'rd='rdcnt-1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r3
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

'rd='rdcnt-1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

rd'=rdcnt' $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

V_{12}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=r4 $\vee$next[i]'=rlseAtPwr))

$\vdash$

V_{13}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r3 $\wedge$ 'rdcnt>1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r3)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=r8)

$\vdash$

V_{14}'

{replace 'V_{13} with its definition}

{get function value of **NextLabel(r3)**}

{substitute for **'rd, 'rdcntt** with **corresponding after-values**}

{get function value of **NextLabel(r3)**}

{get function value of **NextLabel(r3)**}

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt<1)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

Column 12+32*(k-1) for k=1...n, where n is the number of Readers:

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt≥1)

⊢

'V_4 ∧'V_6 ∧'V_7
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt≥1)

{get function value of **NextLabel(r3)**}

{get function value of **NextLabel(r3)**}

{get function value of **NextLabel(r3)**}

{column condition}

≡ {replace 'V_4 'V_6 'V_7 with its definition}

('w.cnt≤1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ ('rdcnt>1⇒ 'rd≥1)
∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt≥1
⊢
('w.cnt<1∨ 'w.cnt=1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ ('rdcnt>1⇒ 'rd≥1)
∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt≥1
⊢
('w.cnt<1∨ 'w.cnt=1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ 'rd≥1
∧ 'w.cnt≥1
⊢
('w.cnt<1∨ 'w.cnt=1)∧¬('w.cnt=1)
⊢
'w.cnt<1
⊢
¬('w.cnt≥1)
⊢
¬ ((pID=k ∧ IsReader) ∧ 'next[k]=r3 ∧ 'rdcnt>1 ∧ 'w.cnt≥1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 13+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=waitAtPwr) {column condition}
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'='next[j]) ∨ (j≠k ∧ next[j]'='next[j])))
proof:
Since all of the variables in all of the invariant clauses are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

Column 14+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rIseAtPwr) {column condition}

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ rd'='rd+1
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_4, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ 'V_8 ∧ 'V_{10}
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

≡ {replace 'V_8 'V_{10} with its definition}

((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ (∃i, i=k ∧'next[i]=rlseAtPwr) ⇒ ('rd=0 ∧'w.cnt<1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ ('rd=0 ∧'w.cnt<1)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ ((('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ 'rd=0
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢

((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
'wt= 0 ∧ 'rd=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))
∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

⊢ {substitute wt' for 'wt}

wt'= 0

⊢

rp_1'

'V_1 ∧ rd'='rd+1

≡

'rd≥0 ∧ rd'='rd+1

⊢ {substitute rd'-1 for 'rd}

rd'-1≥0

$\vdash$

$rd' \geq 1$

$\vdash$

$rd' \geq 0$

$\equiv$

V_1'

$((pID=k \wedge IsReader) \wedge {}'next[k]=rlseAtPwr)$
$\wedge {}'V_{10}$
$\wedge NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)$

$\equiv$

$((pID=k \wedge IsReader) \wedge {}'next[k]=rlseAtPwr)$
$\wedge (\exists i, i=k \wedge {}'next[i]=rlseAtPwr) \Rightarrow ({}'rd=0 \wedge {}'w.cnt<1)$
$\wedge NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)$

$\equiv$

$((pID=k \wedge IsReader) \wedge {}'next[k]=rlseAtPwr)$
$\wedge ({}'rd=0 \wedge {}'w.cnt<1)$
$\wedge NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)$

$\vdash$

$'w.cnt<1 \wedge NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)$

$\vdash$

$w.cnt'<1$

$\vdash$

$\neg(w.cnt'= 1)$

$\vdash$

V_6'

$'V_7 \wedge {}'V_{10}$
$\wedge ((pID=k \wedge IsReader) \wedge {}'next[k]=rlseAtPwr)$
$\wedge NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)$
$\wedge rd'={}'rd+1$

$\equiv$

$({}'rdcnt>1 \Rightarrow {}'rd \geq 1)$
$\wedge (\exists i, i=k \wedge {}'next[i]=rlseAtPwr) \Rightarrow ({}'rd=0 \wedge {}'w.cnt<1)$
$\wedge ((pID=k \wedge IsReader) \wedge {}'next[k]=rlseAtPwr)$
$\wedge NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) \wedge rd'={}'rd+1$

$\vdash$

$({}'rdcnt>1 \Rightarrow {}'rd \geq 1)$
$\wedge ({}'rd=0 \wedge {}'w.cnt<1)$

{replace 'V_{10} with its definition}

{substitute **w.cnt'** for **'w.cnt**}

{replace 'V_7,'V_{10} with its definition}

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)∧ rd'='rd+1

⊢

('rdcnt>1⇒ 'rd≥1) ∧ 'rd=0
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) ∧ rd'='rd+1

⊢ {substitute corresponding after-values for 'rdcnt, 'rd }

(rdcnt'>1 ⇒ rd'-1≥1) ∧ rd'=1

⊢

(rdcnt'>1 ⇒ rd'≥2) ∧ rd'=1

⊢

rdcnt'>1⇒ rd'≥1

≡

V_7'

'V_8 ∧ 'V_{10}
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ rd'='rd+1

≡ {replace 'V_8 'V_{10} with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ (∃i, i=k ∧ 'next[i]=rlseAtPwr))⇒ ('rd=0 ∧'w.cnt<1)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) ∧ rd'='rd+1

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ ('rd=0 ∧'w.cnt<1)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPwr)
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) ∧ rd'='rd+1

⊢

'w.cnt<1 ∧ 'rd=0 ∧ 'wt=0
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set) ∧ rd'='rd+1

⊢ {substitute corresponding after-values for 'rd, 'wt, 'w.cnt }

w.cnt'<1 ∧ rd'=1 ∧ wt'=0

⊢

V_8'

'V_9
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\equiv$ {replace 'V_9 with its definition}

((∃i, i=k∧('next[i]=r3 ∨'next[i]=rlseAtPwr ∨'next[i]=r4 ∨'next[i]=r5 ∨ 'next[i]=r6 ∨'next[i]=rlseAtPm2∨'next[i]=r7)) ⇒ 'rdcnt>0)
∧ 'next[k]=rlseAtPwr
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'rdcnt>0 ∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

rdcnt'>0

$\vdash$

V_9'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j]))

$\vdash$ {get function value of NextLabel(rlseAtPwr)}

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPwr
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

$\vdash$

V_{10}'

'V_{11}
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j]))

$\equiv$ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9))⇒'m.cnt<1
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPwr
∧ (∀j, ((j=k ∧next[j]'=(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

'm.cnt<1 ∧ NC(rdcnt, wt, m.cnt, m.set, w.set, w.cnt)

$\vdash$

m.cnt'<1

$\vdash$

V_{11}'

'V_{13}
∧ rd'='rd+1∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j])))
≡
(∃i, i=k ∧('next[i]=r3∨'next[i]=rlseAtPwr)) ⇒'rd='rdcnt-1
∧ rd'='rd+1∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r3
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j])))
⊢
'rd='rdcnt-1
∧ rd'='rd+1∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))
⊢
rd'=rdcnt' ∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))
⊢
V_{12}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r4 ∨next[i]'=rlseAtPwr))
⊢
V_{13}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧'next[k]=rlseAtPwr)
∧ (∀j, (j=k ∧next[j]'=NextLabel(rlseAtPwr)) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r4) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=r8)

{replace 'V_{13} with its definition}

{get function value of **NextLabel(rlseAtPwr)**}

{substitute for **'rd, 'rdcnt** with **corresponding after-values**}

{get function value of **NextLabel(rlseAtPwr)**}

{get function value of **NextLabel(rlseAtPwr)**}

$\vdash$

V_{14}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$'next[k]=rlseAtPwr)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(rlseAtPwr)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$'next[k]=rlseAtPwr)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(rlseAtPwr)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=w2 $\vee$ next[i]'=w3))

$\vdash$

V_{16}'

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$'next[k]=rlseAtPwr)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(rlseAtPwr)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r4) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=waitAtPm1 $\vee$ next[i]'=waitAtPm2))

$\vdash$

V_{17}'

<u>Column 15+32*(k-1) for k=1...n, where n is the number of Readers:</u>
Given 'INVARIANTS
$\wedge$

{get function value of **NextLabel(rlseAtPwr)**}

{get function value of **NextLabel(rlseAtPwr)**}

{get function value of **NextLabel(rlseAtPwr)**}

{column condition}

⊢

$'V_{11}$

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2∨'next[i]=r3∨'next[i]=rlseAtPwr∨'next[i]=r4∨'next[i]=rlseAtPm2∨'next[i]=r7∨'next[i]=r8∨'next[i]=r9))⇒'m.cnt<1

⊢

'm.cnt<1

⊢

'm.cnt ≤ 0

⊢

¬('m.cnt>0)

⊢

¬ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt>0)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

<u>Column 16+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0) {column condition}

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ m.cnt'='m.cnt+1

∧ (∀j, (j=k ∧next[j]'=NextLabel(r4)) ∨ (j≠k ∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

m.cnt'='m.cnt+1

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0)

⊢

m.cnt'='m.cnt+1

∧ 'm.cnt=0

⊢ {substitute for 'm.cnt with (m.cnt'-1)}

m.cnt'=1

⊢

m.cnt'≤1

≡

V_5'

'V_8

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0)

$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j]))

$\equiv$ {replace 'V_8 with its definition}

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r4 $\wedge$ 'm.cnt=0)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ('next[k]=r4)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t=k$\vee$ t$\neq$k)$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r4) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$
('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t=k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)) $\vee$ (t$\neq$k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r4) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, False$\vee$ (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r4) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists$t, (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r4) $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ *{substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}*

w.cnt'<1$\Rightarrow$ ((wt'=1$\wedge$rd'=0) $\vee$ (rd'$\geq$1$\wedge$wt'=0)$\vee$(rd'=0 $\wedge$wt'=0 $\wedge$($\exists$t, (t$\neq$k$\wedge$ (next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww)))))

$\vdash$

V_8'

'V_9
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r4 $\wedge$ 'm.cnt=0)

$\equiv$ {replace 'V_9 with its definition}

(($\exists$i, i=k$\wedge$('next[i]=r3 $\vee$'next[i]=rlseAtPwr $\vee$'next[i]=r4 $\vee$'next[i]=r5 $\vee$ 'next[i]=r6 $\vee$'next[i]=rlseAtPm2 $\vee$'next[i]=r7)) $\Rightarrow$ 'rdcnt>0)
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r4

$\vdash$

‘rdcnt>0 ∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

⊢

rdcnt’>0

⊢

V_9’

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r4 ∧ ‘m.cnt=0)
∧ (∀j, (j=k ∧next[j]’=NextLabel(r4)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r4)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ‘next[k]=r4
∧ (∀j, ((j=k ∧next[j]’=r5) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧next[i]’=rlseAtPwr)

⊢

V_{10}’

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r4 ∧ ‘m.cnt=0)
∧ (∀j, (j=k ∧next[j]’=NextLabel(r4)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r4)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ‘next[k]=r4
∧ (∀j, (j=k ∧next[j]’=r5) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧(next[i]’=rlseAtPm1∨next[i]’=r2∨next[i]’=r3∨ next[i]’=rlseAtPwr∨next[i]’=r4∨next[i]’=rlseAtPm2∨next[i]’=r7∨next[i]’=r8∨next[i]’=r9))

⊢

V_{11}’

‘V_{12}
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r4 ∧ ‘m.cnt=0)
∧ (∀j, (j=k ∧next[j]’=NextLabel(r4)) ∨ (j≠k ∧next[j]’=‘next[j])))

≡ {replace ‘V_{12} with its definition}

(∃i, i=k ∧(‘next[i]=r1∨‘next[i]=rlseAtPm1∨‘next[i]=r2∨‘next[i]=r4∨‘next[i]=r5∨‘next[i]=r6∨‘next[i]=rlseAtPm2∨‘next[i]=r7∨‘next[i]=r9)) ⇒‘rd=‘rdcnt
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r4 ∧ ‘m.cnt=0)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(r4)) ∨ (j≠k ∧next[j]’=‘next[j])))

$\vdash$

$\quad$ **‘rd=‘rdcnt**

$\quad$ $\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\quad$ **$\wedge$ ‘next[k]=r4**

$\quad$ **$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]’=r5) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))**

$\vdash$

$\quad$ **rd’=rdcnt’**

$\vdash$

$\quad$ **V_{12}’**

$\quad$ **NC(rdcnt, rd, wt, m.set, w.cnt, w.set)**

$\quad$ **$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r4 $\wedge$ ‘m.cnt=0)**

$\quad$ **$\wedge$ ($\forall$j, (j=k $\wedge$next[j]’=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))**

$\vdash$

$\quad$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\quad$ $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]’=r5) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))

$\vdash$

$\quad$ **$\neg$($\exists$i, i=k $\wedge$(next[i]’=r3 $\vee$ next[i]’=rlseAtPwr))**

$\vdash$

$\quad$ **V_{13}’**

$\quad$ **NC(rdcnt, rd, wt, m.set, w.cnt, w.set)**

$\quad$ **$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r4 $\wedge$ ‘m.cnt=0)**

$\quad$ **$\wedge$ ($\forall$j, (j=k $\wedge$next[j]’=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))**

$\vdash$

$\quad$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\quad$ $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]’=r5) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))

$\vdash$

$\quad$ **$\neg$($\exists$i, i=k $\wedge$next[i]’=r8)**

$\vdash$

$\quad$ **V_{14}’**

$\quad$ **NC(rdcnt, rd, wt, m.set, w.cnt, w.set)**

$\quad$ **$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r4 $\wedge$ ‘m.cnt=0)**

$\quad$ **$\wedge$ ($\forall$j, (j=k $\wedge$next[j]’=NextLabel(r4)) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))**

$\vdash$

$\quad$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\quad$ $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]’=r5) $\vee$ (j$\neq$k $\wedge$next[j]’=‘next[j]))

$\vdash$

{get function value of **NextLabel(r4)**}

{get function value of **NextLabel(r4)**}

{get function value of **NextLabel(r4)**}

{get function value of **NextLabel(r4)**}

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r4)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r4)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r5) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r4)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r4)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r5) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

Column 17+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0) {column condition}
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ m.cnt'='m.cnt+1
∧ m.set'=(∃t, (('m.set-{t}) ∧('next[t]=waitAtPm1∨'next[t]=waitAtPm2)))
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4))
∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

proof:
Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

m.cnt'='m.cnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
⊢
m.cnt'='m.cnt+1
∧'m.cnt<0
⊢ {substitute for 'm.cnt with (m.cnt'-1)}
m.cnt'-1<0
⊢
m.cnt'<1
⊢
m.cnt'≤1
≡
V_5'

'V_8
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
≡ {replace 'V_8 with its definition}
('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt=0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k) ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r4)
∧ (∀j, ((j=k∧next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r4)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r4)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r4)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

'V_9
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

≡ {replace 'V_9 with its definition}

((∃i, i=k∧('next[i]=r3 ∨'next[i]=rlseAtPwr ∨'next[i]=r4∨'next[i]=r5∨ 'next[i]=r6 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7)) ⇒ 'rdcnt>0)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ 'next[k]=r4
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

'rdcnt>0∧ NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r5

⊢

rdcnt'>0∧ 'next[k]=r5

⊢

V_9'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))

∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of **NextLabel(r4)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r4
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ m.cnt'='m.cnt+1

⊢

'm.cnt<0 ∧ m.cnt'='m.cnt+1

⊢

m.cnt'< 1

⊢

V_{11}'

'V_{12}
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

≡ {replace 'V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1 ∨'next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9)) ⇒ 'rd='rdcnt
∧ NC(rdcnt, rd, wt, w.cnt, w.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4))∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of **NextLabel(r4)**}

'rd='rdcnt ∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ 'next[k]=r4
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

rd'=rdcnt'

⊢

V_{12}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r4)}

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r4)}

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r4)}

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))

∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=rlseAtPww)
⊢
V_{15}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢ {get function value of NextLabel(r4)}
NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))
⊢
V_{16}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r4 ∧ 'm.cnt<0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r4)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢ {get function value of NextLabel(r4)}
NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'=r5)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))
⊢
V_{17}'

<u>Column 18+32*(k-1) for k=1…n, where n is the number of Readers:</u>
Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r5) {column condition}
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)

$\wedge (\forall j,\ ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_5, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

$'V_8$
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\equiv$ {replace $'V_8$ with its definition}

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge(\exists t,$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists t,$ (t=k$\vee$ t$\neq$k)$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ('next[k]=r5)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$
('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists t,$ (t=k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))$\vee$ (t$\neq$k $\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r5)$\wedge$ $(\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists t,$ False$\vee$ (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r5)$\wedge$ $(\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$('rd=0 $\wedge$'wt=0 $\wedge$ ($\exists t,$ (t$\neq$k$\wedge$ ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))))
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) $\wedge$ ('next[k]=r5)$\wedge$ $(\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$ {substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in $'V_8$}

w.cnt'<1$\Rightarrow$ ((wt'=1$\wedge$rd'=0) $\vee$ (rd'$\geq$1$\wedge$wt'=0)$\vee$(rd'=0 $\wedge$wt'=0 $\wedge(\exists t,$ (t$\neq$k$\wedge$ (next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww)))))

$\vdash$

V_8'

$'V_9$
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(r5)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\equiv$ {replace $'V_9$ with its definition}

((∃i, i=k ∧ (‘next[i]=r3 ∨‘next[i]=rlseAtPwr ∨‘next[i]=r4 ∨‘next[i]=r5 ∨ ‘next[i]=r6∨‘next[i]=rlseAtPm2 ∨‘next[i]=r7)) ⇒ ‘rdcnt>0)
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ‘next[k]=r5
∧ (∀j, ((j=k ∧next[j]’=NextLabel(r5) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r5)**}

‘rdcnt>0
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]’=r6) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

rdcnt’>0∧ (∀j, ((j=k ∧next[j]’=r6) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

V_9’

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r5)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(r5)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r5)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ‘next[k]=r5
∧ (∀j, ((j=k ∧next[j]’=r6) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧next[i]’=rlseAtPwr)

⊢

V_{10}’

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r5)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(r5)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r5)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ‘next[k]=r5
∧ (∀j, (j=k ∧next[j]’=r6) ∨ (j≠k ∧next[j]’=‘next[j]))

⊢

¬(∃i, i=k∧(next[i]’=rlseAtPm1∨next[i]’=r2∨next[i]’=r3∨next[i]’=rlseAtPwr∨ next[i]’=r4∨next[i]’=rlseAtPm2∨next[i]’=r7∨next[i]’=r8∨next[i]’=r9))

⊢

V_{11}’

‘V_{12}
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r5)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r5)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace for V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1 ∨'next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9)) ⇒ 'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r5)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r5)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r5)**}

'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r5
∧ (∀j, ((j=k ∧next[j]'=r6) ∨ (j≠k ∧next[j]'='next[j])))

⊢

rd'=rdcnt'

⊢

V_{12}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r5)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r5)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r5)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r5
∧ (∀j, ((j=k ∧next[j]'=r6) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r5)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r5)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r5)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r5
∧ (∀j, ((j=k ∧next[j]'=r6) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r5)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r5
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r6) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r5)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r5
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r6) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=w2 $\vee$ next[i]'=w3))

$\vdash$

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r5)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r5)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r5
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r6) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=waitAtPm1 $\vee$ next[i]'=waitAtPm2))

$\vdash$

V_{17}'

{get function value of **NextLabel(r5)**}

{get function value of **NextLabel(r5)**}

{get function value of **NextLabel(r5)**}

Column 19+32*(k-1) for k=1...n, where n is the number of Readers:

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt>1) {column condition}

⊢

'V_5

≡

'm.cnt≤1

⊢

¬('m.cnt>1)

⊢

¬((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt>1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 20+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1) {column condition}
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.set)
∧ m.cnt'='m.cnt-1
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r6)) ∨ (j≠k ∧ next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_5

≡

'm.cnt≤1

⊢ {substitute for 'm.cnt with (m.cnt'+1)}

m.cnt'+1≤ 1

⊢

m.cnt'≤ 0

⊢

m.cnt'≤1

≡

V_5'

'V_8
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace for 'V_8 with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r6)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ 'next[k]=r6∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r6) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r6) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ *{substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}*

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

'V_9
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_9 with its definition}

((∃i, i=k ∧ ('next[i]=r3 ∨'next[i]=rlseAtPwr ∨'next[i]=r4∨'next[i]=r5 ∨ 'next[i]=r6 ∨'next[i]=rlseAtPm2∨'next[i]=r7)) ⇒ 'rdcnt>0)
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(r6)}

'rdcnt>0
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

rdcnt'>0∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_9'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(r6)}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ m.cnt'='m.cnt-1

⊢

'm.cnt= 1
∧ m.cnt'='m.cnt-1

⊢

m.cnt'= 0

⊢

m.cnt'< 1

⊢

V_{11}'

'V_{12}
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1 ∨'next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9)) ⇒ 'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)

$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r6)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

'rd='rdcnt
$\wedge$ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r6
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r7) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

rd'=rdcnt'

$\vdash$

V_{12}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt=1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r6)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r6
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r7) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=r3 $\vee$ next[i]'=rlseAtPwr))

$\vdash$

V_{13}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt=1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r6)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r6
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r7) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=r8)

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt=1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r6)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

{get function value of **NextLabel(r6)**}

{get function value of **NextLabel(r6)**}

{get function value of **NextLabel(r6)**}

{get function value of **NextLabel(r6)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r6)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

Column 21+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt<1)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ m.cnt'='m.cnt-1
∧ m.set'='m.set ∪ {k}

{get function value of **NextLabel(r6)**}

{get function value of **NextLabel(r6)**}

{column condition}

$\wedge (\forall j,\ ((j=k \wedge next[j]'=waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

$'V_5$

$\equiv$

'm.cnt≤1

$\vdash$ {substitute for 'm.cnt with (m.cnt'+1)}

m.cnt'+1≤ 1

$\vdash$

m.cnt'≤ 0

$\vdash$

m.cnt'≤1

$\equiv$

V_5'

$'V_8$

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt<1)

$\wedge (\forall j,\ ((j=k \wedge next[j]'=waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\equiv$ {replace 'V_8 with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt=1)

$\wedge (\forall j, ((j=k \wedge next[j]'=waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ ('next[k]=r6)

$\wedge (\forall j, ((j=k \wedge next[j]'= waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ('next[k]=r6)∧ $(\forall j, ((j=k \wedge next[j]'=waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

$\wedge$ NC(rdcnt, rd, wt, w.cnt, w.set) $\wedge$ ('next[k]=r6)∧ $(\forall j, ((j=k \wedge next[j]'=waitAtPm2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨ (‘rd=0 ∧‘wt=0 ∧ (∃t, False∨ (t≠k∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, w.cnt, w.set) ∧ (‘next[k]=r6)∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨(‘rd=0 ∧‘wt=0 ∧ (∃t, (t≠k∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, w.cnt, w.set) ∧ (‘next[k]=r6)∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {substitute for ‘wt, ‘rd, ‘w.cnt, ‘next with corresponding after-values in ‘V_8}

w.cnt’<1⇒ ((wt’=1∧rd’=0) ∨ (rd’≥1∧wt’=0)∨(rd’=0 ∧wt’=0 ∧(∃t, (t≠k∧ (next[t]’=rlseAtPwr ∨next[t]’=rlseAtPww)))))

⊢

V_8’

‘V_9
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r6 ∧ ‘m.cnt<1)
∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

≡ {replace ‘V_9 with its definition}

((∃i, i=k ∧ (‘next[i]=r3 ∨‘next[i]=rlseAtPwr ∨‘next[i]=r4∨‘next[i]=r5 ∨ ‘next[i]=r6∨‘next[i]=rlseAtPm2 ∨‘next[i]=r7)) ⇒ ‘rdcnt>0)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ‘next[k]=r6
∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

‘rdcnt>0
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

rdcnt’>0∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

V_9’

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r6 ∧ ‘m.cnt<1)
∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ‘next[k]=r6
∧ (∀j, ((j=k ∧next[j]’=waitAtPm2) ∨ (j≠k ∧next[j]’=‘next[j])))
∧¬(∃i, i=k ∧next[i]’=rlseAtPwr)

⊢

V_{10}’

((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt<1)
∧ m.cnt'='m.cnt-1

⊢

'm.cnt<1
∧ m.cnt'='m.cnt-1

⊢

m.cnt'<0

⊢

m.cnt'< 1

⊢

V_{11}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPm2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=waitAtPm2) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=r1∨next[i]'=rlseAtPm1∨next[i]'=r2∨next[i]'=r4∨next[i]'=r5∨next[i]'=r6∨next[i]'=rlseAtPm2∨next[i]'=r7∨next[i]'=r9))

⊢

V_{12}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPm2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ 'next[k]=r6
∧ (∀j, ((j=k ∧next[j]'=waitAtPm2) ∨ (j≠k ∧next[j]'='next[j])))
∧¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r6 ∧ 'm.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPm2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, w.cnt, w.set)

$\wedge$ 'next[k]=r6$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))
$\wedge\neg$($\exists$i, i=k $\wedge$next[i]'=r8)

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, w.cnt, w.set)
$\wedge$ 'next[k]=r6
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))
$\wedge\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

NC(rdcnt, rd, wt, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, w.cnt, w.set)
$\wedge$ 'next[k]=r6
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))
$\wedge\neg$($\exists$i, i=k $\wedge$(next[i]'=w2 $\vee$ next[i]'=w3))

$\vdash$

V_{16}'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r6 $\wedge$ 'm.cnt<1)
$\wedge$ m.cnt'='m.cnt-1
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

'm.cnt< 1 $\wedge$ m.cnt'='m.cnt-1
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

m.cnt'< 0 $\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'= waitAtPm2) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

V_{17}'

Column 22+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=waitAtPm2) {column condition}
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧ next[j]'='next[j]) ∨ (j≠k ∧ next[j]'='next[j])))
proof:
Since all of the variables in all of the invariant clauses are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

Column 23+32*(k-1) for k=1...n, where n is the number of Readers:
Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2) {column condition}
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))
proof:
Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_5, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.
For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:
'V_8
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))
≡ {replace 'V_8 with its definition}
('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ 'next[k]=rlseAtPm2∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, Falsе∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ 'next[k]=rlseAtPm2 ∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ 'next[k]=rlseAtPm2 ∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0) ∨ (rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

'V_9

∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)

∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

≡ {replace 'V_9 with its definition}

((∃i, i=k ∧ ('next[i]=r3 ∨'next[i]=rlseAtPwr ∨'next[i]=r4∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7)) ⇒ 'rdcnt>0)

∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)

∧ 'next[k]=rlseAtPm2

∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPm2)}

'rdcnt>0

∧ NC(rdcnt, rd, wt, w.cnt, w.set, m.cnt, m.set)

∧ 'next[k]=rlseAtPm2

∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

rdcnt'>0∧ (∀j, ((j=k ∧ next[j]'=r7) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

V_9'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)

∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPm2)}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ 'next[k]=rlseAtPm2

∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

$`V_{11}$
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9)) ⇒'m.cnt<1
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPm2)}

'm.cnt<1
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

m.cnt'<1 ∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

$`V_{11}$'

$`V_{12}$
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

≡ {replace 'V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1 ∨'next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9))⇒ 'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPm2)}

'rd='rdcnt
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

rd'=rdcnt'

⊢

$`V_{12}$'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

{get function value of **NextLabel(rlseAtPm2)**}

{get function value of **NextLabel(rlseAtPm2)**}

{get function value of **NextLabel(rlseAtPm2)**}

⊢ {get function value of NextLabel(rlseAtPm2)}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=rlseAtPm2)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(rlseAtPm2)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPm2)}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=rlseAtPm2
∧ (∀j, ((j=k ∧next[j]'=r7) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 24+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7) {column condition}
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ rdcnt'='rdcnt-1
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r7)) ∨ (j≠k ∧ next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_4, V_5, V_6 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ 'V_9
∧ rdcnt'='rdcnt-1

⊢

((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧(∃i, 'next[i]=r7 ⇒ 'rdcnt>0)
∧ rdcnt'='rdcnt-1

⊢

'rdcnt>0∧ rdcnt'='rdcnt-1

⊢ {substitute for 'rdcnt with (rdcnt'+1)}

rdcnt'+1>0

⊢

rdcnt'>-1

⊢

rdcnt'≥0

≡

V_3'

'V_7
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ rdcnt'='rdcnt-1

≡

'rdcnt>1⇒ 'rd ≥ 0
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ rdcnt'='rdcnt-1

⊢ {substitute for 'rdcnt, 'rd with (rdcnt'+1), rd'}

rdcnt'+1>1⇒ rd' ≥ 0

⊢

rdcnt'>0⇒ rd' ≥ 0
∧ rdcnt'>1⇒ rdcnt'>0

⊢

rdcnt'>1⇒ rd' ≥ 0

≡

V_7'

'V_8
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r7)) ∨ (j≠k ∧ next[j]'='next[j]))

≡ {replace 'V_8 with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r7)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r7)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

(‘rd=0 ∧‘wt=0 ∧ (∃t, (t=k ∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))∨ (t≠k ∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r7)∧ (∀j, ((j=k ∧ next[j]’=NextLabel(r7)) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨ (‘rd=0 ∧‘wt=0 ∧ (∃t, False∨ (t≠k∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r7)∧ (∀j, ((j=k ∧ next[j]’=NextLabel(r7)) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢

(‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0) ∨ (‘rd=0 ∧‘wt=0 ∧ (∃t, (t≠k∧ (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww))))))
∧ NC(rd, wt, w.cnt, w.set, m.cnt, m.set) ∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r7)∧ (∀j, ((j=k ∧ next[j]’=NextLabel(r7)) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢ *{substitute for ‘wt, ‘rd, ‘w.cnt, ‘next with corresponding after-values in V_8}*

w.cnt’<1⇒ ((wt’=1∧rd’=0)∨(rd’≥1∧wt’=0)∨(rd’=0 ∧wt’=0 ∧(∃t, (t≠k∧ (next[t]’=rlseAtPwr ∨next[t]’=rlseAtPww)))))

⊢

V_8’

NC(rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ rdcnt’=‘rdcnt-1
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r7)
∧ (∀j, ((j=k ∧ next[j]’=NextLabel(r7)) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rd, wt, w.cnt, w.set, m.cnt, m.set)
∧ rdcnt’=‘rdcnt-1
∧ ‘next[k]=r7
∧ (∀j, ((j=k ∧ next[j]’=r8) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧(next[i]’=r3 ∨ next[i]’=rlseAtPwr ∨ next[i]’=r4∨ next[i]’=r5 ∨ next[i]’=r6∨ next[i]’=rlseAtPm2∨next[i]’=r7))

⊢

V_9’

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=r7)
∧ (∀j, ((j=k ∧ next[j]’=NextLabel(r7)) ∨ (j≠k ∧ next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ‘next[k]=r7
∧ (∀j, ((j=k ∧next[j]’=r8) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧next[i]’=rlseAtPwr)

⊢

V_{10}’

$'V_{11}$
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r7)) ∨ (j≠k ∧ next[j]'='next[j])))

≡ {replace $'V_{11}$ with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9)) ⇒'m.cnt<1
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r7
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

'm.cnt<1
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

m.cnt'<1∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{11}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k∧(next[i]'=r1∨next[i]'=rlseAtPm∨next[i]'=r2∨next[i]'=r4∨next[i]'=r5∨next[i]'=r6∨next[i]'=rlseAtPm2∨next[i]'=r7∨next[i]'=r9))

⊢

V_{12}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

'V_{12}
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ rdcnt'='rdcnt-1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_{12} with its definition}

(∃i, i=k ∧('next[i]=r1∨'next[i]=rlseAtPm1∨'next[i]=r2∨'next[i]=r4
∨'next[i]=r5 ∨'next[i]=r6 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r9))⇒'rd='rdcnt
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ 'next[k]=r7
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

'rd='rdcnt
∧ NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {substitute for **'rd, 'rdcntt** with **corresponding after-values**}

rd'=rdcnt'+1 ∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

V_{14}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r7)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r7)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(r7)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=r8) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 25+32*(k-1) for k=1…n, where n is the number of Readers:</u>

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0∧'w.cnt>0)

⊢

'V_4 ∧'V_6∧'V_{14}
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0∧'w.cnt>0)

≡

('w.cnt≤1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ (∃i, i=k ∧ 'next[i]=r8) ⇒ 'rd='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0∧'w.cnt>0)

⊢

('w.cnt<1∨ 'w.cnt=1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ (∃i, 'next[i]=r8) ⇒ 'rd='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0∧'w.cnt>0)

⊢

('w.cnt<1∨ 'w.cnt=1)
∧ 'w.cnt=1⇒ ('wt=0∧'rd=0)
∧ 'rd=1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0∧'w.cnt>0)

⊢

('w.cnt<1∨ 'w.cnt=1)

{get function value of **NextLabel(r7)**}

{column condition}

{replace 'V_4 'V_6 'V_{14} with its definition}

$\wedge \neg$('w.cnt=1)

$\vdash$

'w.cnt<1

$\vdash$

$\neg$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0$\wedge$'w.cnt>0)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 26+32*(k-1) for k=1...n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0) {column condition}

$\wedge$ NC(rdcnt, wt, w.set, m.cnt, m.set)

$\wedge$ rd'='rd-1

$\wedge$ w.cnt'='w.cnt+1

$\wedge$ ($\forall$j, ((j=k $\wedge$ next[j]'=NextLabel(r8)) $\vee$ (j$\neq$k $\wedge$ next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)

$\wedge$ 'V_{14} $\wedge$ 'V_8

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\equiv$

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)

$\wedge$ ($\exists$i, i=k $\wedge$ 'next[i]=r8) $\Rightarrow$ 'rd='rdcnt+1

$\wedge$ ('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

('next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)

$\wedge$ 'rd='rdcnt+1

$\wedge$ ('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

('next[k]=r8 $\wedge$ 'w.cnt=0)

$\wedge$ 'rd=1

$\wedge$ ('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0) $\vee$ ('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

'rd$\geq$1 $\wedge$ 'wt= 0$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

'wt= 0$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)

$\vdash$

wt'= 0

$\equiv$

rp_1'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ 'V_{14}
$\wedge$ rd'='rd-1

$\equiv$

('next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ ($\exists$i, i=k $\wedge$ 'next[i]=r8) $\Rightarrow$ 'rd='rdcnt+1
$\wedge$ rd'='rd-1

$\vdash$

'rdcnt=0 $\wedge$ 'rd='rdcnt+1$\wedge$ rd'='rd-1

$\vdash$

'rd=1 $\wedge$ rd'='rd-1

$\vdash$

rd'=0

$\vdash$

rd'$\geq$0

$\equiv$

V_1'

((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ w.cnt'='w.cnt+1

$\vdash$

'w.cnt=0$\wedge$ w.cnt'='w.cnt+1

$\vdash$

w.cnt'=1

$\vdash$

w.cnt'$\leq$ 1

$\equiv$

V_4'

'V_8 $\wedge$ 'V_{14}
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ rd'='rd-1

{substitute for **'wt** with **wt'**}

{substitute for **'rd** with **rd'-1**}

{substitute for **'w.cnt** with **w.cnt'-1**}

∧ w.cnt'='w.cnt-1

≡

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ (∃i, i=k ∧ 'next[i]=r8) ⇒ 'rd='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ rd'='rd-1 ∧ w.cnt'='w.cnt-1

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ 'rd='rdcnt+1
∧ 'rdcnt=0 ∧ 'w.cnt=0
∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ 'rd=1
∧ 'w.cnt=0
∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1

⊢

('rd=1 ∧ 'wt=0)
∧ 'w.cnt=0
∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1

⊢ {substitute for 'rd, 'wt, 'w.cnt with corresponding after-values}

(rd'=0 ∧wt'=0) ∧ w.cnt'=1

⊢

V_6'

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ NC(rdcnt, wt, m.cnt, m.set, w.set)

⊢

'rdcnt=0

⊢ {substitute for 'rdcnt with rdcnt'}

rdcnt'=0

⊢

¬(rdcnt'> 1)

⊢

V_7'

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ w.cnt'='w.cnt-1

⊢

'w.cnt=0 ∧ w.cnt'='w.cnt-1

⊢ {substitute for **'w.cnt** with **w.cnt'**}

w.cnt'=1

⊢

¬(w.cnt'< 1)

⊢

V_8'

NC(rdcnt, wt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, w.set, m.cnt, m.set)
∧ 'next[k]=r8
∧ (∀j, ((j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧ (next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, wt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set, w.set)
∧ 'next[k]=r8
∧ (∀j, ((j=k ∧next[j]'=r9) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

'V_{11}
∧ NC(rdcnt, wt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr

$\vee$'next[i]=r4 $\vee$'next[i]=rlseAtPm2 $\vee$'next[i]=r7 $\vee$'next[i]=r8 $\vee$'next[i]=r9))$\Rightarrow$'m.cnt<1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)
$\wedge$ 'next[k]=r8
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r8)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(r8)**}

'm.cnt<1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r9) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

m.cnt'<1$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r9) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

V_{11}'

'V_{14}
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set)
$\wedge$ rd'='rd-1
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r8)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\equiv$ {replace 'V_{14} with its definition}

($\exists$i, i=k $\wedge$'next[i]=r8) $\Rightarrow$'rd='rdcnt+1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set) $\wedge$ rd'='rd-1
$\wedge$ 'next[k]=r8
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(r8)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(r8)**}

'rd='rdcnt+1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.set) $\wedge$ rd'='rd-1
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r9) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {substitute for **'rd, 'rdcntt** with **corresponding after-values**}

rd'=rdcnt'

$\vdash$

V_{12}'

NC(rdcnt, wt, m.cnt, m.set, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt=0)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r8)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=r8) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg(\exists i,\ i=k \wedge (next[i]'=r3 \vee next[i]'=rlseAtPwr))$

$\vdash$

V_{13}'

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge$ (($pID=k \wedge IsReader$) $\wedge$ '$next[k]=r8 \wedge$ '$rdcnt=0 \wedge$ '$w.cnt=0$)
$\wedge (\forall j,\ (j=k \wedge next[j]'=NextLabel(r8)) \vee (j\neq k \wedge next[j]'='next[j]))$

$\vdash$

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge (\forall j,\ ((j=k \wedge next[j]'=r9) \vee (j\neq k \wedge next[j]'='next[j]))$)

$\vdash$

$\neg(\exists i,\ i=k \wedge next[i]'=r8)$

$\vdash$

V_{14}'

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge$ (($pID=k \wedge IsReader$) $\wedge$ '$next[k]=r8 \wedge$ '$rdcnt=0 \wedge$ '$w.cnt=0$)
$\wedge (\forall j,\ (j=k \wedge next[j]'=NextLabel(r8)) \vee (j\neq k \wedge next[j]'='next[j]))$

$\vdash$

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge (\forall j,\ ((j=k \wedge next[j]'=r9) \vee (j\neq k \wedge next[j]'='next[j]))$)

$\vdash$

$\neg(\exists i,\ i=k \wedge next[i]'=rlseAtPww)$

$\vdash$

V_{15}'

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge$ (($pID=k \wedge IsReader$) $\wedge$ '$next[k]=r8 \wedge$ '$rdcnt=0 \wedge$ '$w.cnt=0$)
$\wedge (\forall j,\ (j=k \wedge next[j]'=NextLabel(r8)) \vee (j\neq k \wedge next[j]'='next[j]))$

$\vdash$

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge (\forall j,\ ((j=k \wedge next[j]'=r9) \vee (j\neq k \wedge next[j]'='next[j]))$)

$\vdash$

$\neg(\exists i,\ i=k \wedge (next[i]'=w2 \vee next[i]'=w3))$

$\vdash$

V_{16}'

$NC(rdcnt, wt, m.cnt, m.set, w.set)$
$\wedge$ (($pID=k \wedge IsReader$) $\wedge$ '$next[k]=r8 \wedge$ '$rdcnt=0 \wedge$ '$w.cnt=0$)

{get function value of **NextLabel(r8)**}

{get function value of **NextLabel(r8)**}

{get function value of **NextLabel(r8)**}

∧ (∀j, (j=k ∧next[j]'=NextLabel(r8)) ∨ (j≠k ∧next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set, w.set)

∧ (∀j, ((j=k ∧next[j]'=r9) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 27+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0) {column condition}

∧ NC(rdcnt, wt, m.cnt, m.set)

∧ rd'='rd-1

∧ w.cnt'='w.cnt+1

∧ w.set'=(∃t, ('w.set-{t}) ∧'next[t]=waitAtPww)

∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))

∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)

∧ 'V_{14} ∧ 'V_8

∧ NC(rdcnt, wt, m.cnt, m.set)

≡

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)

∧ (∃i, i=k ∧ 'next[i]=r8) ⇒ 'rd='rdcnt+1

∧ ('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, wt, m.cnt, m.set)

⊢

('next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)

∧ 'rd='rdcnt+1

∧ ('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, wt, m.cnt, m.set)

⊢

('next[k]=r8 ∧ 'w.cnt<0)

∧ 'rd=1

∧ (‘w.cnt<1⇒ ((‘wt=1∧‘rd=0)∨(‘rd≥1∧‘wt=0)∨(‘rd=0 ∧‘wt=0 ∧(∃t, (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww)))))
∧ NC(rdcnt, wt, m.cnt, m.set)

⊢

‘rd≥1 ∧ ‘wt= 0 ∧ NC(rdcnt, wt, m.cnt, m.set)

⊢

‘wt= 0 ∧ NC(rdcnt, wt, m.cnt, m.set)

⊢ {substitute for ‘wt with wt’}

wt’= 0

≡

rp_1’

((pID=k ∧ IsReader) ∧ ‘next[k]=r8 ∧ ‘rdcnt=0 ∧ ‘w.cnt<0)
∧ ‘V_{14}
∧ rd’=‘rd-1

≡

(‘next[k]=r8 ∧ ‘rdcnt=0 ∧ ‘w.cnt<0)
∧ (∃i, i=k ∧ ‘next[i]=r8) ⇒ ‘rd=‘rdcnt+1
∧ rd’=‘rd-1

⊢

‘rdcnt=0 ∧ ‘rd=‘rdcnt+1 ∧ rd’=‘rd-1

⊢

‘rd=1 ∧ rd’=‘rd-1

⊢ {substitute for ‘rd with rd’-1}

rd’=0

⊢

rd’≥0

≡

V_3’

((pID=k ∧ IsReader) ∧ ‘next[k]=r8 ∧ ‘rdcnt=0 ∧ ‘w.cnt<0)
∧ w.cnt’=‘w.cnt+1

⊢

‘w.cnt<0 ∧ w.cnt’=‘w.cnt+1

⊢ *{substitute for ‘w.cnt with w.cnt’-1}*

w.cnt’<1

⊢

w.cnt’≤ 1

≡

V_4’

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ w.cnt'='w.cnt-1

⊢

'w.cnt<0 ∧ w.cnt'='w.cnt+1

⊢ *{substitute for 'w.cnt with corresponding after-values}*

w.cnt'<1

⊢

¬(w.cnt'= 1)

⊢

V_6'

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ NC(rdcnt, wt, m.cnt, m.set)

⊢

'rdcnt=0

⊢ *{substitute for 'rdcnt with rdcnt'}*

rdcnt'=0

⊢

¬(rdcnt'> 1)

⊢

V_7'

'V_8
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ 'V_{14}
∧ NC(rdcnt, wt, m.cnt, m.set)
∧ rd'='rd-1
∧ w.cnt'='w.cnt+1
∧ w.set'=(∃t, ('w.set-{t}) ∧'next[t]=waitAtPww)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

≡ *{replace 'V_8, 'V_{14} with its definition}*

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ (∃i, i=k ∧ 'next[i]=r8) ⇒ 'rd='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ NC(rdcnt, wt, m.cnt, m.set) ∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1 ∧ w.set'=(∃t, ('w.set-{t}) ∧'next[t]=waitAtPww)
∧ (∀j, (j=k∧next[j]'=NextLabel(r8)) ∨ (j≠k∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k∧ j∉('w.set-w.set')∧next[j]'='next[j]))))

⊢

('rd≥1∧'wt=0)
∧ 'rd=1
∧ 'next[k]=r8 ∧ 'w.cnt<0
∧ NC(rdcnt, wt, m.cnt, m.set) ∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1 ∧ w.set'=(∃t, ('w.set-{t}) ∧'next[t]=waitAtPww)
∧ (∀j, (j=k∧next[j]'=NextLabel(r8)) ∨ (j≠k∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k∧ j∉('w.set-w.set')∧next[j]'='next[j]))))

⊢

'wt=0
∧ 'rd=1
∧ 'next[k]=r8 ∧ 'w.cnt<0
∧ NC(rdcnt, wt, m.cnt, m.set) ∧ rd'='rd-1 ∧ w.cnt'='w.cnt+1 ∧ w.set'=(∃t, ('w.set-{t}) ∧'next[t]=waitAtPww)
∧ (∀j, (j=k∧next[j]'=NextLabel(r8))∨ (j≠k∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ *{substitute for 'w.cnt, 'wt, 'rd}*

w.cnt'<1
∧ (rd'=0 ∧wt'=0 ∧(∃t, t∈('w.set-w.set') ∧next[t]'=rlseAtPww)

⊢

V_8'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r8)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ 'next[k]=r8
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

$\neg(\exists i,\ i=k \wedge next[i]'=rlseAtPwr)$

⊢

V_{10}'

$'V_{11}$
$\wedge$ NC(rdcnt, wt, m.cnt, m.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt<0)
$\wedge$ ($\forall j$, (j=k $\wedge$ next[j]'=NextLabel(r8))
$\vee$ (j≠k $\wedge$ j∈('w.set-w.set') $\wedge$'next[j]=waitAtPww $\wedge$ next[j]'=rlseAtPww)$\vee$ (j≠k $\wedge$ j∉('w.set-w.set')$\wedge$next[j]'='next[j])))

≡ *{replace 'V_{11} with its definition}*

($\exists i,\ i=k \wedge$('next[i]=rlseAtPm1$\vee$'next[i]=r2 $\vee$'next[i]=r3$\vee$'next[i]=rlseAtPwr
$\vee$'next[i]=r4 $\vee$'next[i]=rlseAtPm2 $\vee$'next[i]=r7 $\vee$'next[i]=r8 $\vee$'next[i]=r9)) $\Rightarrow$'m.cnt<1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set)
$\wedge$ 'next[k]=r8
$\wedge$ ($\forall j$, (j=k $\wedge$ next[j]'=NextLabel(r8)) $\vee$ (j≠k $\wedge$ j∈('w.set-w.set') $\wedge$'next[j]=waitAtPww $\wedge$ next[j]'=rlseAtPww)$\vee$ (j≠k $\wedge$ j∉('w.set-w.set')$\wedge$next[j]'='next[j])))

⊢ *{get function value of **NextLabel(r8)**}*

'm.cnt<1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set)
$\wedge$ ($\forall j$, (j=k $\wedge$ next[j]'=r9) $\vee$ (j≠k $\wedge$ j∈('w.set-w.set') $\wedge$'next[j]=waitAtPww $\wedge$ next[j]'=rlseAtPww)$\vee$ (j≠k $\wedge$ j∉('w.set-w.set')$\wedge$next[j]'='next[j])))

⊢

m.cnt'<1

⊢

V_{11}'

$'V_{14}$
$\wedge$ NC(rdcnt, wt, m.cnt, m.set)
$\wedge$ rd'='rd-1
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt=0 $\wedge$ 'w.cnt<0)
$\wedge$ ($\forall j$, (j=k $\wedge$ next[j]'=NextLabel(r8))
$\vee$ (j≠k $\wedge$ j∈('w.set-w.set') $\wedge$'next[j]=waitAtPww $\wedge$ next[j]'=rlseAtPww)$\vee$ (j≠k $\wedge$ j∉('w.set-w.set')$\wedge$next[j]'='next[j])))

⊢ *{replace 'V_{14} with its definition}*

($\exists i,\ i=k \wedge$'next[i]=r8) $\Rightarrow$'rd='rdcnt+1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set)
$\wedge$ rd'='rd-1
$\wedge$ 'next[k]=r8
$\wedge$ ($\forall j$, (j=k $\wedge$ next[j]'=NextLabel(r8))$\vee$ (j≠k $\wedge$ j∈('w.set-w.set') $\wedge$'next[j]=waitAtPww $\wedge$ next[j]'=rlseAtPww)$\vee$ (j≠k $\wedge$ j∉('w.set-w.set')$\wedge$next[j]'='next[j])))

⊢ *{get function value of **NextLabel(r8)**}*

'rd='rdcnt+1

∧ NC(rdcnt, wt, m.cnt, m.set) ∧ rd'='rd-1
∧ (∀j, (j=k ∧ **next[j]'=r9**) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ *{substitute for 'rd, 'rdcntt with corresponding after-values}*

rd'=rdcnt'

⊢

V_{12}'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt=0 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8))
∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ j∈('w.set-w.set') ∧'next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

Column 28+32*(k-1) for k=1...n, where n is the number of Readers:

Given 'INVARIANTS
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt<0) {column condition}

⊢

'V_3

≡

'rdcnt≥0

⊢

¬ ('rdcnt<0)

⊢

¬((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt<0)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 29+32*(k-1) for k=1…n, where n is the number of Readers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0) {column condition}

∧ NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)

∧ rd'='rd-1

∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

proof:

Since all of the variables in the invariant clauses rp_2, V_2, V_3, V_4, V_5 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

'V_4∧'V_6∧'V_8∧'V_{14}

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

≡

'w.cnt ≤1

∧ 'w.cnt=1⇒ 'wt=0∧'rd=0

∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))

∧ (∃i, i=k∧'next[i]=r8) ⇒ 'rd='rdcnt+1

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt ≤1

∧ 'w.cnt=1⇒ 'wt=0∧'rd=0

∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))

∧ 'rd='rdcnt+1

∧ ('rdcnt>0)

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt ≤1

∧ 'w.cnt=1⇒ 'wt=0∧'rd=0

∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))

∧ 'rd>1

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt<1

∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))

$\wedge$ ‘rd>1

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

‘rd>1 $\wedge$ ‘wt= 0 $\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

‘wt= 0 $\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$ {substitute for ‘wt with wt’}

wt’= 0

$\vdash$

rp_1’

((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r8 $\wedge$ ‘rdcnt>0)

$\wedge$‘V_{14}

$\wedge$ rd’=‘rd-1

$\vdash$

(‘next[k]=r8 $\wedge$ ‘rdcnt>0)

$\wedge$ ($\exists$i, i=k$\wedge$‘next[i]=r8) $\Rightarrow$ ‘rd=‘rdcnt+1

$\wedge$ rd’=‘rd-1

$\vdash$

(‘rdcnt>0)

$\wedge$ ‘rd=‘rdcnt+1

$\wedge$ rd’=‘rd-1

$\vdash$

‘rd>1 $\wedge$ rd’=‘rd-1

$\vdash$ {substitute for ‘rd with rd’}

rd’> 0

$\vdash$

V_1’

‘$V_4$$\wedge$‘$V_6$$\wedge$‘$V_8$$\wedge$‘$V_{14}$

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r8 $\wedge$ ‘rdcnt>0)

$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\equiv$

‘w.cnt $\leq$1

$\wedge$ ‘w.cnt=1$\Rightarrow$ ‘wt=0$\wedge$‘rd=0

$\wedge$ ‘w.cnt<1$\Rightarrow$ ((‘wt=1$\wedge$‘rd=0)$\vee$(‘rd$\geq$1$\wedge$‘wt=0)$\vee$(‘rd=0 $\wedge$‘wt=0 $\wedge$($\exists$t, (‘next[t]=rlseAtPwr $\vee$‘next[t]=rlseAtPww))))

$\wedge$ ($\exists$i, i=k$\wedge$‘next[i]=r8) $\Rightarrow$ ‘rd=‘rdcnt+1

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ ‘next[k]=r8 $\wedge$ ‘rdcnt>0)

∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt ≤1
∧ 'w.cnt=1⇒ 'wt=0∧'rd=0
∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ **'rd='rdcnt+1**
∧ **('rdcnt>0)**
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt ≤1
∧ 'w.cnt=1⇒ 'wt=0∧'rd=0
∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ **'rd>1**
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt<1
∧ **'rd>1**
∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢

'w.cnt<1 ∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

⊢ {substitute for 'wt with **wt'**}

w.cnt'< 1

⊢

¬(w.cnt'=1)

⊢

V_6'

'V_4∧'V_6∧'V_8∧'V_{14}
∧ **((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)**
∧ **rd'='rd-1**
∧ **NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)**

≡

'w.cnt ≤1
∧ 'w.cnt=1⇒ 'wt=0∧'rd=0
∧ 'w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ (∃i, i=k∧'next[i]=r8) ⇒ 'rd='rdcnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ rd'='rd-1∧ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'rd>1 $\wedge$ ('rdcnt>0)
$\wedge$ rd'='rd-1$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$ {substitute for 'rd, 'rdcnt with rd'+1, rdcnt'}

rd'>0 $\wedge$ rdcnt'>0

$\vdash$

rd'$\geq$1 $\wedge$ rdcnt'>0

$\vdash$

rdcnt'>0$\Rightarrow$ rd'$\geq$1 $\wedge$ rdcnt'>1 $\Rightarrow$ rdcnt'>0

$\vdash$

rdcnt'>1 $\Rightarrow$ rd'$\geq$1

$\equiv$

V_7'

'$V_4\wedge$'$V_6\wedge$'$V_8\wedge$'V_{14}
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt>0)
$\wedge$ rd'='rd-1
$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\equiv$

'w.cnt $\leq$1
$\wedge$ 'w.cnt=1$\Rightarrow$ 'wt=0$\wedge$'rd=0
$\wedge$ 'w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0)$\vee$('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rIseAtPwr $\vee$'next[t]=rIseAtPww))))
$\wedge$ ($\exists$i, i=k$\wedge$'next[i]=r8) $\Rightarrow$ 'rd='rdcnt+1
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r8 $\wedge$ 'rdcnt>0)
$\wedge$ rd'='rd-1$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'w.cnt <1
$\wedge$'rd>1 $\wedge$'wt=0
$\wedge$ rd'='rd-1$\wedge$ NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)

$\vdash$ {substitute for 'rd, 'wt, 'w.cnt with corresponding after-values}

w.cnt' <1
$\wedge$ rd'>0 $\wedge$wt'=0

$\vdash$

w.cnt' <1
$\wedge$ (rd' $\geq$0 $\wedge$wt'=0)

$\vdash$

V_8'

((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

⊢

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ('next[k]=r8)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ('next[k]=r8)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j]))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ 'next[k]=r8
∧ (∀j, ((j=k ∧next[j]'=r9) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

'V_{11}
∧ NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

≡ {replace for'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9))⇒'m.cnt<1
NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ('next[k]=r8)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j]))

⊢ {get function value of **NextLabel(r8)**}

$$
\begin{array}{l}
\text{`m.cnt<1} \\
\text{NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)} \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=r9}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
\text{m.cnt'<1} \wedge (\forall j, (j=k \wedge \text{next[j]'=r9}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
V_{11}\text{'} \\
\\
\text{`}V_{14} \\
\wedge \text{ NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)} \\
\wedge \text{rd'=`rd-1} \\
\wedge ((\text{pID=k} \wedge \text{IsReader}) \wedge \text{`next[k]=r8} \wedge \text{`rdcnt>0}) \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=NextLabel(r8)}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\equiv \\
(\exists i, i=k \wedge \text{`next[i]=r8}) \Rightarrow \text{`rd=`rdcnt+1} \\
\wedge \text{NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)} \\
\wedge \text{rd'=`rd-1} \\
\wedge \text{`next[k]=r8} \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=NextLabel(r8)}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
\text{`rd=`rdcnt+1} \\
\wedge \text{NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)} \wedge \text{rd'=`rd-1} \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=r9}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
\text{rd'=rdcnt'} \\
\vdash \\
V_{12}\text{'} \\
\\
\text{NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)} \\
\wedge ((\text{pID=k} \wedge \text{IsReader}) \wedge \text{`next[k]=r8} \wedge \text{`rdcnt>0}) \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=NextLabel(r8)}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
\text{NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)} \\
\wedge (\forall j, (j=k \wedge \text{next[j]'=r9}) \vee (j \neq k \wedge \text{next[j]'=`next[j]}))) \\
\vdash \\
\neg(\exists i, i=k \wedge (\text{next[i]'=r3} \vee \text{next[i]'=rlseAtPwr})) \\
\vdash \\
V_{13}\text{'}
\end{array}
$$

{replace 'V_{14} with its definition}

{get function value of **NextLabel(r8)**}

{substitute for **'rd, 'rdcntt** with **corresponding after-values**}

{get function value of **NextLabel(r8)**}

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

⊢

V_{15}'

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, wt, w.cnt, w.set, m.cnt, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r8 ∧ 'rdcnt>0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(r8)) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

NC(rdcnt, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, (j=k ∧ next[j]'=r9) ∨ (j≠k ∧ next[j]'='next[j])))

{get function value of **NextLabel(r8)**}

{get function value of **NextLabel(r8)**}

{get function value of **NextLabel(r8)**}

{get function value of **NextLabel(r8)**}

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 30+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt>0) {column condition}

⊢

'V_{11}

≡ {replace 'V_{11} with its definition}

(∃i, i=k ∧('next[i]=rlseAtPm1∨'next[i]=r2 ∨'next[i]=r3∨'next[i]=rlseAtPwr
∨'next[i]=r4 ∨'next[i]=rlseAtPm2 ∨'next[i]=r7 ∨'next[i]=r8 ∨'next[i]=r9))⇒'m.cnt<1

⊢

'm.cnt<1

⊢

'm.cnt ≤ 0

⊢

¬('m.cnt>0)

⊢

¬((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt>0)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

<u>Column 31+32*(k-1) for k=1...n, where n is the number of Readers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0) {column condition}

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ m.cnt'='m.cnt+1

∧ (∀j, (j=k∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

m.cnt'='m.cnt+1

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)

⊢

m.cnt'='m.cnt+1

∧ 'm.cnt=0

⊢ {substitute for 'm.cnt with (m.cnt'-1)}

m.cnt'=1

⊢

m.cnt'≤1

≡

V_5'

'V_8

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧(∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_8 with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

∧ ('next[k]=r9)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))

∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧(∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r9)
∧ (∀j, (j=k ∧ next[j]'=END) ∨ (j≠k ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, wt, w.cnt, w.set, m.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧(∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, w.cnt, w.set, m.set)
∧ 'next[k]=r9
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r9
∧ (∀j, (j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬ (∃i, i=k∧(next[i]'=rlseAtPm1∨next[i]'=r2∨next[i]'=r3∨next[i]'=rlseAtPwr ∨next[i]'=r4∨next[i]'=rlseAtPm2∨next[i]'=r7∨ next[i]'=r8∨ next[i]'=r9))

⊢

V_{11}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(r9)**}
NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r9
∧ (∀j, (j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
∧¬(∃i, i=k ∧(next[i]'=r1 ∨next[i]'=rlseAtPm1∨ next[i]'=r2 ∨ next[i]'=r4 ∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨ next[i]'=r9))
⊢
V_{12}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(r9)**}
NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r9
∧ (∀j, (j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))
⊢
V_{13}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt=0)
∧ (∀j, (j=k ∧next[j]'=NextLabel(r9)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(r9)**}
NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ 'next[k]=r9
∧ (∀j, (j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=r8)
⊢
V_{14}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r9 $\wedge$ 'm.cnt=0)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r9)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r9
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=END) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r9 $\wedge$ 'm.cnt=0)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r9)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r9
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=END) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=w2 $\vee$ next[i]'=w3))

$\vdash$

V_{16}'

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r9 $\wedge$ 'm.cnt=0)
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=NextLabel(r9)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
$\wedge$ 'next[k]=r9
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=END) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=waitAtPm1 $\vee$ next[i]'=waitAtPm2))

$\vdash$

V_{17}'

<u>Column 32+32*(k-1) for k=1...n, where n is the number of Readers:</u>
Prove: INVARIANTS'
Given 'INVARIANTS
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=r9 $\wedge$ 'm.cnt<0)

{get function value of **NextLabel(r9)**}

{get function value of **NextLabel(r9)**}

{get function value of **NextLabel(r9)**}

{column condition}

∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ m.cnt'='m.cnt+1
∧ m.set'=(∃t, ('m.set-{t}) ∧'next[t]=waitAtPm2)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses rp_1, rp_2, V_1, V_2, V_3, V_4, V_6, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses which have variables changed in the state change, we will prove them one at a time as below:

m.cnt'='m.cnt+1
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
⊢
m.cnt'='m.cnt+1
∧'m.cnt<0
⊢ {substitute for 'm.cnt with (m.cnt'-1)}
m.cnt'-1<0
⊢
m.cnt'<1
⊢
m.cnt'≤1
≡
V_5'

'V_8
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ NC(rdcnt, rd, wt, w.cnt, w.set)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
≡ {replace 'V_8 with its definition}
('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r1 ∧ 'm.cnt=1)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set)
∧ ('next[k]=r9)

∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨
('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))
∧ NC(rdcnt, rd, wt, m.set, w.cnt, w.set) ∧ ('next[k]=r9)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ ('next[k]=r9)
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))

∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j]))) ⊢ ¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7)) ⊢ V_9'

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0) ∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2))) ∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j]))) ⊢ {get function value of **NextLabel(r9)**}

NC(rdcnt, rd, wt, w.cnt, w.set)∧ 'next[k]=r9 ∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2))) ∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j]))) ⊢ ¬(∃i, i=k ∧next[i]'=rlseAtPwr) ⊢ V_{10}'

((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0) ∧ m.cnt'='m.cnt+1 ⊢ 'm.cnt<0 ∧ m.cnt'='m.cnt+1 ⊢ m.cnt'< 1 ⊢ V_{11}'

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0) ∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2))) ∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j]))) ⊢ {get function value of **NextLabel(r9)**}

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r9 ∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2))) ∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r1 ∨next[i]'=rlseAtPm1∨ next[i]'=r2 ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨ next[i]'=r9))

⊢

V_{12}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ 'next[k]=r9
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'
NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r9
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))

⊢ {get function value of NextLabel(r9)}

NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r9
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=rlseAtPww)
⊢
V_{15}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢ {get function value of NextLabel(r9)}
NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r9
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))
⊢
V_{16}'

NC(rdcnt, rd, wt, w.cnt, w.set)
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=r9 ∧ 'm.cnt<0)
∧ (∀j, ((j=k∧ next[j]'=NextLabel(r9)) ∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢ {get function value of NextLabel(r9)}
NC(rdcnt, rd, wt, w.cnt, w.set) ∧ 'next[k]=r9
∧ (∀j, ((j=k ∧ next[j]'=END)∨(j≠k ∧ j∈('m.set-m.set')∧(('next[j]=waitAtPm1 ∧ next[j]'=rlseAtPm1)∨('next[j]=waitAtPm2∧ next[j]'=rlseAtPm2)))
∨(j≠k ∧ j∉('m.set-m.set') ∧ next[j]'='next[j])))
⊢
∧¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))
⊢
V_{17}'

Columns for Writer processes:
Column 1+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:
Given 'INVARIANTS

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt>1) {column condition}

⊢

'V_4

≡

'w.cnt≤1

⊢

¬('m.cnt>1)

⊢

¬ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt>1)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 2+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1) {column condition}

∧ wt'='wt+1

∧ w.cnt'='w.cnt-1

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses V_1, V_3, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

'V_6

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)

∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

≡

('w.cnt=1⇒ ('wt=0 ∧ 'rd=0))

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)

∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢

'wt= 0 ∧ 'rd=0

⊢

'rd=0

⊢ {substitute for 'rd with rd'}

rd'= 0

⊢

rp_t'

$`V_6$
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt=1)
$\wedge$ wt'=`wt+1
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.set)

$\equiv$

(`w.cnt=1$\Rightarrow$ (`wt=0 $\wedge$ `rd=0))
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt=1)
$\wedge$ wt'=`wt+1 $\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.set)

$\vdash$

`wt= 0 $\wedge$ `rd=0 $\wedge$ wt'=`wt+1

$\vdash$

`wt=0 $\wedge$ wt'=`wt+1

$\vdash$

wt'= 1

$\vdash$

rp_2' $\wedge$ V_2'

((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt=1)
$\wedge$ w.cnt'=`w.cnt-1

$\vdash$

`w.cnt=1 $\wedge$ w.cnt'=`w.cnt-1

$\vdash$

w.cnt'=0

$\vdash$

V_4'

((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt=1)
$\wedge$ w.cnt'=`w.cnt-1

$\vdash$

`w.cnt=1 $\wedge$ w.cnt'=`w.cnt-1

$\vdash$

w.cnt'=0

$\vdash$

$\neg$(w.cnt'=1)

$\vdash$

V_6'

$`V_6$
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt=1)

{substitute for 'wt with wt'-1}

∧ wt'='wt+1
∧ w.cnt'='w.cnt-1
∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

≡

('w.cnt=1⇒ ('wt=0 ∧ 'rd=0))
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ wt'='wt+1 ∧ w.cnt'='w.cnt-1∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢

'wt= 0 ∧ 'rd=0
∧ wt'='wt+1 ∧ w.cnt'='w.cnt-1∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢ {substitute for 'wt , 'rd, 'w.cnt with corresponding after-values}

wt'= 1∧ rd'=0
∧ w.cnt'=0

⊢

V_8'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(w1)}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rIseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rIseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(w1)}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rIseAtPwr)

⊢

V_{10}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w1)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=rlseAtPm1∨next[i]'=r2∨next[i]'=r3∨next[i]'=rlseAtPwr∨next[i]'=r4∨next[i]'=rlseAtPm2∨next[i]'=r7∨next[i]'=r8∨next[i]'=r9))

⊢

V_{11}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w1)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))

⊢

V_{12}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w1)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃l, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)

$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(w1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

NC(rdcnt, rd, m.set, m.cnt, w.set)
$\wedge$ ('next[k]=w1)
$\wedge (\forall j, ((j=k \wedge next[j]'=w2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge next[i]'=r8)$

$\vdash$

V_{14}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt=1)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(w1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

NC(rdcnt, rd, m.set, m.cnt, w.set)
$\wedge$ ('next[k]=w1)
$\wedge (\forall j, ((j=k \wedge next[j]'=w2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

$\neg(\exists i, i=k \wedge next[i]'=rlseAtPww)$

$\vdash$

V_{15}'

'V_6
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt=1)
$\wedge$ wt'='wt+1
$\wedge$ w.cnt'='w.cnt-1
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.set)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(w1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\equiv$

('w.cnt=1$\Rightarrow$ ('wt=0 $\wedge$ 'rd=0))
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt=1)
$\wedge$ wt'='wt+1 $\wedge$ w.cnt'='w.cnt-1$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.set)
$\wedge (\forall j, ((j=k \wedge next[j]'=NextLabel(w1)) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

'wt= 0 $\wedge$ 'rd=0
$\wedge$ wt'='wt+1 $\wedge$ w.cnt'='w.cnt-1$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.set)
$\wedge (\forall j, ((j=k \wedge next[j]'=w2) \vee (j\neq k \wedge next[j]'='next[j])))$

$\vdash$

wt'= 1$\wedge$ w.cnt'=0$\wedge (\forall j, ((j=k \wedge next[j]'=w2) \vee (j\neq k \wedge next[j]'='next[j])))$

{get function value of **NextLabel(w1)**}

{get function value of **NextLabel(w1)**}

{substitute for **'wt , 'w.cnt** with **corresponding after-values**}

⊢

V_{16}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w1)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 3+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:</u>

Prove: INVARIANTS'
Given 'INVARIANTS
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1) {column condition}
∧ w.cnt'='w.cnt-1
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
proof:
Since all of the variables in the invariant clauses **rp1, rp2,** V_1, V_2, V_3, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.
For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ w.cnt'='w.cnt-1

⊢

'w.cnt<1∧ w.cnt'='w.cnt-1

⊢

w.cnt'<0

⊢

V_4'

((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ w.cnt'='w.cnt-1

⊢

'w.cnt<1∧ w.cnt'='w.cnt-1

$\vdash$

$w.cnt' < 0$

$\vdash$

$\neg(w.cnt' = 1)$

$\vdash$

V_6'

$`V_8$
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt<1)
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set)
$\wedge$ w.cnt'=`w.cnt-1
$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'=`next[j])))

$\equiv$ {replace $`V_8$ with its definition}

`w.cnt<1 $\Rightarrow$ ((`wt=1$\wedge$`rd=0)$\vee$(`rd$\geq$1$\wedge$`wt=0)$\vee$(`rd=0 $\wedge$`wt=0 $\wedge$($\exists$t,(`next[t]=rlseAtPwr$\vee$`next[t]=rlseAtPww))))
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt<1)
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set) $\wedge$ w.cnt'=`w.cnt-1$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'=`next[j])))

$\vdash$

`w.cnt<1
$\wedge$((`wt=1$\wedge$`rd=0)$\vee$(`rd$\geq$1$\wedge$`wt=0)$\vee$(`rd=0 $\wedge$`wt=0 $\wedge$($\exists$t,(`next[t]=rlseAtPwr$\vee$`next[t]=rlseAtPww))))
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt<1)
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set) $\wedge$ w.cnt'=`w.cnt-1$\wedge$ ($\forall$j, (j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'=`next[j])))

$\vdash$ {substitute for **'w.cnt, 'wt, 'rd, 'next** with **corresponding after-values}**

$w.cnt' < 0$
$\wedge$ ((wt'=1$\wedge$rd'=0)$\vee$(rd'$\geq$1$\wedge$wt'=0)$\vee$(rd'=0 $\wedge$wt'=0 $\wedge$($\exists$t,(next[t]'=rlseAtPwr $\vee$next[t]'=rlseAtPww))))

$\vdash$

V_8'

NC(rdcnt, rd, wt, m.cnt, m.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ `next[k]=w1 $\wedge$ `w.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'=`next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt)
$\wedge$ (`next[k]=w1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'=`next[j])))

$\vdash$

$\neg(\exists i, i=k \wedge (next[i]'=r3 \vee next[i]'=rlseAtPwr \vee next[i]'=r4 \vee next[i]'=r5 \vee next[i]'=r6 \vee next[i]'=rlseAtPm2 \vee next[i]'=r7))$

$\vdash$

V_9'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, m.set, m.cnt)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=rlseAtPwr)
⊢
V_{10}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, m.set, m.cnt)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=rlseAtPm1∨next[i]'=r2∨next[i]'=r3∨next[i]'=rlseAtPwr∨next[i]'=r4∨next[i]'=rlseAtPm2 ∨ next[i]'=r7∨next[i]'=r8∨next[i]'=r9))
⊢
V_{11}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
NC(rdcnt, rd, wt, m.set, m.cnt)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))
⊢
V_{12}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt)
$\wedge$ (**'next[k]=w1**)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=r3 $\vee$ next[i]'=rlseAtPwr))

$\vdash$

V_{13}'

NC(rdcnt, rd, wt, m.cnt, m.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt)
$\wedge$ (**'next[k]=w1**)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=r8)

$\vdash$

V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt)
$\wedge$ (**'next[k]=w1**)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

NC(rdcnt, rd, wt, m.cnt, m.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w1 $\wedge$ 'w.cnt<1)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=waitAtPww) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt)
$\wedge$ (**'next[k]=w1**)

∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))

⊢

(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt<1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))

⊢

NC(rdcnt, rd, wt, m.set, m.cnt)
∧ ('next[k]=w1)
∧ (∀j, ((j=k ∧next[j]'=waitAtPww) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 4+9*(k-1) for k=1...n, where n is the number of Writers and M is the number of Readers and Writers:</u>

Prove: INVARIANTS'
Given 'INVARIANTS
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=waitAtPww) {column condition}
∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=pID ∧ next[j]'='next[j]) ∨ (j≠pID ∧ next[j]'='next[j])))
proof:
Since all of the variables in all of the invariant clauses are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

<u>Column 5+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:</u>

Prove: INVARIANTS'
Given 'INVARIANTS
∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww) {column condition}
∧ wt'='wt+1
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))
proof:
Since all of the variables in the invariant clauses V_1, V_3, V_4, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.
For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:
'V_{15} ∧'V_8

$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=rlseAtPww)
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\equiv$

($\exists$i, i=k $\wedge$'next[i]=rlseAtPww) $\Rightarrow$ ('wt=0 $\wedge$'w.cnt<1)
$\wedge$ ('w.cnt<1$\Rightarrow$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww)))))
$\wedge$ 'next[k]=rlseAtPww)
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'wt= 0 $\wedge$ 'w.cnt<1
$\wedge$ (('wt=1$\wedge$'rd=0) $\vee$ ('rd$\geq$1$\wedge$'wt=0)$\vee$('rd=0 $\wedge$'wt=0 $\wedge$($\exists$t, ('next[t]=rlseAtPwr $\vee$'next[t]=rlseAtPww))))
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'rd=0 $\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\vdash$ {substitute for 'rd with rd'}

rd'=0

$\vdash$

rp_1'

'V_{15}
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=rlseAtPww)
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
$\wedge$ wt'='wt+1

$\equiv$

($\exists$i, i=k $\wedge$'next[i]=rlseAtPww) $\Rightarrow$ ('wt=0 $\wedge$'w.cnt<1)
$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=rlseAtPww)
$\wedge$ wt'='wt+1 $\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'wt= 0 $\wedge$ 'w.cnt<1
$\wedge$ wt'='wt+1 $\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

$\vdash$

'wt= 0 $\wedge$ wt'='wt+1

$\vdash$ {substitute for 'wt with wt'-1}

wt'= 1

$\vdash$

rp_2' $\wedge$ V_2'

'V_{15}
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=rlseAtPww)
$\wedge$ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

≡

(∃i, i=k ∧'next[i]=rlseAtPww) ⇒ ('wt=0 ∧'w.cnt<1)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

⊢

'wt= 0 ∧ 'w.cnt<1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

⊢

w.cnt'<1

⊢

¬(w.cnt'=1)

⊢

V_6'

'V_{15} ∧'V_8
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ wt'='wt+1
∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

≡

(∃i, i=k ∧'next[i]=rlseAtPww) ⇒ ('wt=0 ∧'w.cnt<1)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ 'next[k]=rlseAtPww
∧ wt'='wt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

⊢

'wt= 0 ∧ 'w.cnt<1
∧ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ 'next[k]=rlseAtPww
∧ wt'='wt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

⊢

'wt= 0 ∧ 'w.cnt<1 ∧ 'rd=0
∧ wt'='wt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

⊢ {substitute for 'wt , 'rd, 'w.cnt with corresponding after-values}

wt'=1∧ rd'=0
∧ w.cnt'<1

⊢

V_8'

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(rlseAtPww)**}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ (‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=w2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧(next[i]’=r3 ∨ next[i]’=rlseAtPwr ∨ next[i]’=r4∨ next[i]’=r5 ∨ next[i]’=r6 ∨ next[i]’=rlseAtPm2∨next[i]’=r7))

⊢

V_9’

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(rlseAtPww)**}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ (‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=w2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧next[i]’=rlseAtPwr)

⊢

V_{10}’

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(rlseAtPww)**}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ (‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=w2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧(next[i]’=rlseAtPm1 ∨ next[i]’=r2∨ next[i]’=r3∨ next[i]’=rlseAtPwr∨ next[i]’=r4∨ next[i]’=rlseAtPm2 ∨ next[i]’=r7∨next[i]’=r8∨next[i]’=r9))

⊢

V_{11}’

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢ {get function value of **NextLabel(rlseAtPww)**}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ (‘next[k]=rlseAtPww)

∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))

⊢

V_{12}'

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPww)}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPww)}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=r8)

⊢

V_{14}'

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(rlseAtPww)}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPww)

$\vdash$

V_{15}'

'V_{15} ∧'V_8
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ wt'='wt+1
∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

≡

(∃i, i=k ∧'next[i]=rlseAtPww) ⇒ ('wt=0 ∧'w.cnt<1)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ 'next[k]=rlseAtPww
∧ wt'='wt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

'wt= 0 ∧ 'w.cnt<1
∧ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))
∧ 'next[k]=rlseAtPww
∧ wt'='wt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set) ∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(rlseAtPww)**}

'wt= 0 ∧ 'w.cnt<1
∧ wt'='wt+1
∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$ {substitute for **'wt , 'w.cnt** with **corresponding after-values}**

wt'=1 ∧ w.cnt'<1∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

V_{16}'

NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(rlseAtPww)) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$ {get function value of **NextLabel(rlseAtPww)**}

NC(rdcnt, rd, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=rlseAtPww)
∧ (∀j, ((j=k ∧next[j]'=w2) ∨ (j≠k ∧next[j]'='next[j])))

$\vdash$

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

$\vdash$

V_{17}'

<u>Column 6+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:</u>

Prove: INVARIANTS'

Given 'INVARIANTS

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2) {column condition}

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses **rp1, rp2, V_1, V_2, V_3, V_4, V_5, V_6,V_7** are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

'V_8

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)

∧ NC(rdcnt, rd, m.cnt, m.set, w.cnt, w.set)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

≡ {replace 'V_8 with its definition}

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=rlseAtPww)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧ (∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

∧ ('next[k]=rlseAtPww)

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, (t=k∨ t≠k)∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPww)∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨

('rd=0 ∧'wt=0 ∧ (∃t, (t=k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))∨ (t≠k ∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPww)∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧ (∃t, False∨ (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPww)∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢

('w.cnt<1⇒ (('wt=1∧'rd=0)∨('rd≥1∧'wt=0) ∨('rd=0 ∧'wt=0 ∧ (∃t, (t≠k∧ ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww))))))

∧ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set) ∧ ('next[k]=rlseAtPww)∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ *{substitute for 'wt, 'rd, 'w.cnt, 'next with corresponding after-values in 'V_8}*

w.cnt'<1⇒ ((wt'=1∧rd'=0) ∨ (rd'≥1∧wt'=0)∨(rd'=0 ∧wt'=0 ∧(∃t, (t≠k∧ (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww)))))

⊢

V_8'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w2)**}

NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w2)**}

NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w2)**}

NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r3∨ next[i]'=rlseAtPwr∨ next[i]'=r4 ∨ next[i]'=rlseAtPm2 ∨ next[i]'=r7∨next[i]'=r8∨next[i]'=r9))

⊢

V_{11}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w2)**}
NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rIseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rIseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))
⊢
V_{12}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w2)**}
NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rIseAtPwr))
⊢
V_{13}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w2)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w2)**}
NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
∧ ('next[k]=w2)
∧ (∀j, ((j=k ∧next[j]'=w3) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=r8)
⊢
V_{14}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)

$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'='NextLabel(w2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
$\wedge$ ('next[k]=w2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=w3) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$next[i]'=rlseAtPww)

$\vdash$

V_{15}'

'V_{16}
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w2)
$\wedge$ NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(w2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\equiv$

($\exists$i, i=k $\wedge$('next[i]=w2 $\vee$'next[i]=w3)) $\Rightarrow$ ('wt=1 $\wedge$'w.cnt<1)
$\wedge$ ('next[k]=w2)
$\wedge$ NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(w2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

'wt= 1 $\wedge$ 'w.cnt<1
NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
$\wedge$ ('next[k]=w2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=w3) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

wt'=1 $\wedge$ w.cnt'<1$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=w3) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

V_{16}'

NC(rdcnt, rd, wt, m.cnt, m.set, w.cnt, w.set)
$\wedge$ ((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=NextLabel(w2)) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

NC(rdcnt, rd, wt, m.set, m.cnt, w.cnt, w.set)
$\wedge$ ('next[k]=w2)
$\wedge$ ($\forall$j, ((j=k $\wedge$next[j]'=w3) $\vee$ (j$\neq$k $\wedge$next[j]'='next[j])))

$\vdash$

$\neg$($\exists$i, i=k $\wedge$(next[i]'=waitAtPm1 $\vee$ next[i]'=waitAtPm2))

{get function value of **NextLabel(w2)**}

{get function value of **NextLabel(w2)**}

{substitute for **'wt , 'w.cnt** with **corresponding after-values**}

{get function value of **NextLabel(w2)**}

⊢

V_{17}'

Column 7+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:

Given 'INVARIANTS

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt>0) {column condition}

⊢

'V_{16}

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt>0)

≡

(∃i, i=k ∧ ('next[j]=w2 ∨'next[j]=w3))⇒ ('wt=1 ∧'w.cnt<1) ∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt>0)

⊢

'wt=1 ∧ 'w.cnt<1

⊢

'w.cnt<1

⊢

'w.cnt≤0

⊢

¬('w.cnt>0)

⊢

¬((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt>0)

Here, we derived the negation of the column condition, which proves any state changes described in this column will not occur.

Column 8+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:

Prove: INVARIANTS'

Given 'INVARIANTS

∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0) {column condition}

∧ wt'='wt-1

∧ w.cnt'='w.cnt+1

∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))

proof:

Since all of the variables in the invariant clauses V_1, V_3, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.

For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

'V_{16} ∧'V_8

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)

∧ wt'='wt-1

∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

≡

(∃i, i=k ∧ (‘next[i]=w2 ∨ ‘next[i]=w3)) ⇒ (‘wt=1 ∧‘w.cnt<1)
∧ (‘w.cnt<1⇒ ((‘wt=1∧‘rd=0) ∨ (‘rd≥1∧‘wt=0)∨(‘rd=0 ∧‘wt=0 ∧(∃t, (‘next[t]=rlseAtPwr ∨‘next[t]=rlseAtPww)))))
∧ ((pID=k ∧ IsReader) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0) ∧ wt’=‘wt-1 ∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢

‘wt=1 ∧‘w.cnt<1∧‘rd=0

⊢

‘rd=0

⊢ {substitute for ‘rd with rd’}

rd’= 0

⊢

rp_1’

‘V_{16}
∧ wt’=‘wt-1
∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

≡

(∃i, i=k ∧ (‘next[i]=w2 ∨ ‘next[i]=w3)) ⇒ (‘wt=1 ∧‘w.cnt<1)
∧ wt’=‘wt-1 ∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢

‘wt=1 ∧‘w.cnt<1∧ wt’=‘wt-1

⊢

‘wt=1 ∧ wt’=‘wt-1

⊢ {substitute for ‘wt with wt’+1}

wt’= 0

⊢

rp_2’ ∧ V_2’

((pID=k ∧ IsWriter) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0)
∧ w.cnt’=‘w.cnt+1

⊢

‘w.cnt=0∧ w.cnt’=‘w.cnt+1

⊢

w.cnt’=1

⊢

V_4’

‘V_{16} ∧‘V_8
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0)

∧ wt'='wt-1
∧ w.cnt'='w.cnt+1
∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

≡

(∃i, i=k ∧ ('next[i]=w2 ∨ 'next[i]=w3)) ⇒ ('wt=1 ∧'w.cnt<1)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0)∨('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))
∧ ((pID=k ∧ IsReader) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ wt'='wt-1 ∧ w.cnt'='w.cnt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢

'wt=1 ∧'w.cnt=0∧'rd=0∧ wt'='wt-1 ∧ w.cnt'='w.cnt+1∧ NC(rdcnt, rd, m.cnt, m.set, w.set)

⊢ {substitute for 'rd, 'wt, 'w.cnt with corresponding after-values}

wt'=0 ∧w.cnt'=1∧rd'=0

⊢

V_6'

((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ w.cnt'='w.cnt+1

⊢

'w.cnt= 0∧ w.cnt'='w.cnt-1

⊢ {substitute for 'w.cnt with 'w.cnt+1}

w.cnt'=1

⊢

¬ (w.cnt'<1)

⊢

V_8'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(w3)}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr ∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, m.cnt, m.set, w.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w3)**}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧next[i]'=rlseAtPwr)
⊢
V_{10}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w3)**}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=rlseAtPm1∨next[i]'=r2∨next[i]'=r3∨next[i]'=rlseAtPwr∨ next[i]'=r4 ∨ next[i]'=rlseAtPm2∨next[i]'=r7∨next[i]'=r8∨next[i]'=r9))
⊢
V_{11}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w3)**}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))
⊢
¬(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))
⊢
V_{12}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w1 ∧ 'w.cnt=1)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w1)) ∨ (j≠k ∧next[j]'='next[j])))
⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ (‘next[k]=w1)
∧ (∀j, ((j=k ∧next[j]’=w2) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧(next[i]’=r3 ∨ next[i]’=rlseAtPwr))

⊢

V_{13}’

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(w3)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ (‘next[k]=w3)
∧ (∀j, ((j=k ∧next[j]’=END) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

¬(∃i, i=k ∧next[i]’=r8)

⊢

V_{14}’

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(w3)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ (‘next[k]=w3)
∧ (∀j, ((j=k ∧next[j]’=END) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

(∃i, i=k ∧next[i]’=rlseAtPww)

⊢

V_{15}’

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ ‘next[k]=w3 ∧ ‘w.cnt=0)
∧ (∀j, ((j=k ∧next[j]’=NextLabel(w3)) ∨ (j≠k ∧next[j]’=‘next[j])))

⊢

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ (‘next[k]=w3)
∧ (∀j, ((j=k ∧next[j]’=END) ∨ (j≠k ∧next[j]’=‘next[j])))

{get function value of **NextLabel(w3)**}

{get function value of **NextLabel(w3)**}

{get function value of **NextLabel(w3)**}

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, m.cnt, m.set, w.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt=0)
∧ (∀j, ((j=k ∧next[j]'=NextLabel(w3)) ∨ (j≠k ∧next[j]'='next[j])))

⊢ {get function value of NextLabel(w3)}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, ((j=k ∧next[j]'=END) ∨ (j≠k ∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'

<u>Column 9+9*(k-1) for k=n+1...M, where n is the number of Writers and M is the number of Readers and Writers:</u>

Prove: INVARIANTS'
Given 'INVARIANTS
∧ NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0) {column condition}
∧ wt'='wt-1
∧ w.cnt'='w.cnt+1
∧ w.set'=(∃t, (('w.set-{t}) ∧('next[t]=waitAtPww∨'next[t]=waitAtPwr)))
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

proof:
Since all of the variables in the invariant clauses V_1, V_3, V_5, V_7 are not changed, the truth value of them will not change, hence, they will be true with the after-values of the variables.
For the invariant clauses in which the variables changed in the state change, we will prove them one at a time as below:

'V_{16} ∧'V_8
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ wt'='wt-1
∧ NC(rdcnt, rd, m.cnt, m.set)
≡
(∃i, i=k ∧ ('next[i]=w2 ∨ 'next[i]=w3)) ⇒ ('wt=1 ∧'w.cnt<1)
∧ ('w.cnt<1⇒ (('wt=1∧'rd=0) ∨ ('rd≥1∧'wt=0) ∨ ('rd=0 ∧'wt=0 ∧(∃t, ('next[t]=rlseAtPwr ∨'next[t]=rlseAtPww)))))

$\wedge$ ((pID=k $\wedge$ IsReader) $\wedge$ 'next[k]=w3 $\wedge$ 'w.cnt<0)
$\wedge$ wt'='wt-1 $\wedge$ NC(rdcnt, rd, m.cnt, m.set)

$\vdash$

'wt=1 $\wedge$'w.cnt<1$\wedge$'rd=0

$\vdash$

'rd=0

$\vdash$

rd'= 0

$\vdash$

rp_1'

'V_{16}
$\wedge$ wt'='wt-1
$\wedge$ NC(rdcnt, rd, m.cnt, m.set)

$\equiv$

($\exists$i, i=k $\wedge$ ('next[i]=w2 $\vee$ 'next[i]=w3)) $\Rightarrow$ ('wt=1 $\wedge$'w.cnt<1)
$\wedge$ wt'='wt-1 $\wedge$ NC(rdcnt, rd, m.cnt, m.set)

$\vdash$

'wt=1 $\wedge$'w.cnt<1$\wedge$ wt'='wt-1

$\vdash$

'wt=1$\wedge$ wt'='wt-1

$\vdash$

wt'= 0

$\vdash$

rp_2' $\wedge$ V_2'

((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w3 $\wedge$ 'w.cnt<0)
$\wedge$ w.cnt'='w.cnt+1

$\vdash$

'w.cnt<0$\wedge$ w.cnt'='w.cnt+1

$\vdash$

w.cnt'<1

$\vdash$

V_4'

((pID=k $\wedge$ IsWriter) $\wedge$ 'next[k]=w3 $\wedge$ 'w.cnt<0)
$\wedge$ w.cnt'='w.cnt+1

$\vdash$

'w.cnt<0$\wedge$ w.cnt'='w.cnt+1

{substitute for 'rd with rd'}

{substitute for 'wt with wt'+1}

⊢

w.cnt'<1

⊢

¬(w.cnt'=1)

⊢

V_6'

'V_{16}

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)

∧ NC(rdcnt, rd, m.cnt, m.set)

∧ wt'='wt-1

∧ w.cnt'='w.cnt+1

∧ w.set'=(∃t, (('w.set-{t}) ∧('next[t]=waitAtPww∨'next[t]=waitAtPwr)))

∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))

∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))

∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

≡

(∃i, i=k ∧ ('next[i]=w2 ∨ 'next[i]=w3)) ⇒ ('wt=1 ∧'w.cnt<1)

∧ ((pID=k ∧ IsReader) ∧ 'next[k]=w3 ∧ 'w.cnt<0)

∧ NC(rdcnt, rd, m.cnt, m.set)

∧ wt'='wt-1 ∧ w.cnt'='w.cnt+1

∧ w.set'=(∃t, ('w.set-{t}) ∧('next[t]=waitAtPww∨'next[t]=waitAtPwr))

∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))

∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))

∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

'wt=1 ∧'w.cnt<0∧'rd=0

∧ NC(rdcnt, rd, m.cnt, m.set) ∧ wt'='wt-1 ∧ w.cnt'='w.cnt+1 ∧ w.set'=(∃t, ('w.set-{t}) ∧('next[t]=waitAtPww∨'next[t]=waitAtPwr))

∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))

∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))

∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {substitute for 'wt, 'rd, 'w.cnt, with corresponding after-values}

'w.cnt<1∧rd'=0 ∧wt'=0 ∧(∃t, (next[t]'=rlseAtPwr ∨next[t]'=rlseAtPww))

⊢

V_8'

NC(rdcnt, rd, m.cnt, m.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)

∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))

∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))

⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr∨next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6 ∨ next[i]'=rlseAtPm2∨next[i]'=r7))

⊢

V_9'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

(∃i, i=k ∧next[i]'=rlseAtPwr)

⊢

V_{10}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))

∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j]))))

⊢

(∃i, i=k ∧(next[i]'=rlseAtPm1∨next[i]'=r2∨ next[i]'=r3∨ next[i]'=rlseAtPwr∨ next[i]'=r4∨next[i]'=rlseAtPm2∨ next[i]'=r7∨next[i]'=r8∨next[i]'=r9))

⊢

V_{11}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

(∃i, i=k ∧(next[i]'=r1∨ next[i]'=rlseAtPm1 ∨ next[i]'=r2∨ next[i]'=r4∨ next[i]'=r5 ∨ next[i]'=r6∨next[i]'=rlseAtPm2 ∨ next[i]'=r7 ∨next[i]'=r9))

⊢

V_{12}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of **NextLabel(w3)**}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

(∃i, i=k ∧(next[i]'=r3 ∨ next[i]'=rlseAtPwr))

⊢

V_{13}'

NC(rdcnt, rd, m.cnt, m.set)

∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))
⊢ {get function value of NextLabel(w3)}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))
⊢
(∃i, i=k ∧next[i]'=r8)
⊢
V_{14}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))
⊢ {get function value of NextLabel(w3)}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))
⊢
(∃i, i=k ∧next[i]'=rlseAtPww)
⊢
V_{15}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))
⊢ {get function value of NextLabel(w3)}
NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)

∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=w2 ∨ next[i]'=w3))

⊢

V_{16}'

NC(rdcnt, rd, m.cnt, m.set)
∧ ((pID=k ∧ IsWriter) ∧ 'next[k]=w3 ∧ 'w.cnt<0)
∧ (∀j, (j=k ∧ next[j]'=NextLabel(w3))
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[j]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢ {get function value of NextLabel(w3)}

NC(rdcnt, rd, m.set, m.cnt, w.set)
∧ ('next[k]=w3)
∧ (∀j, (j=k ∧ next[j]'=END)
∨ (j≠k ∧ j∈('w.set-w.set')∧(('next[j]=waitAtPww ∧ next[j]'=rlseAtPww)∨('next[t]=waitAtPwr∧ next[j]'=rlseAtPwr))
∨ (j≠k ∧ j∉('w.set-w.set')∧next[j]'='next[j])))

⊢

¬(∃i, i=k ∧(next[i]'=waitAtPm1 ∨ next[i]'=waitAtPm2))

⊢

V_{17}'