DOI 10.1007/s00165-014-0303-1
BCS © 2014
Formal Aspects of Computing (2015) 27: 103–131

Formal Aspects of Computing

Verification of distributed systems with the axiomatic system of MSVL

Qian Ma, Zhenhua Duan, Nan Zhang and Xiaobing Wang

Institute of Computing Theory and Technology, Xidian University, No. 2 South Tai Bai Road, P.O. Box 177, Xi'an 710071, China

Abstract. Since distributed systems are inherently concurrent and asynchronous, it is a challenge for us to verify distributed systems. MSVL is a useful temporal logic programming language and its axiomatic system has been established. However, the axiomatic system of MSVL lacks mechanisms to manage asynchronous communication, which makes it cannot deal with distributed systems. Thus, to verify distributed systems with MSVL in a deductive way, this paper is motivated to extend the axiomatic system of MSVL with new axioms for asynchronous communication. To this end, firstly we formalize state axioms regarding asynchronous communication commands and then prove the soundness and completeness. Further, to demonstrate how the extended axiomatic system of MSVL works for distributed systems, we apply it to the well-known Ricart–Agrawala (RA) algorithm, which is a distributed mutual exclusion algorithm and has an infinite state space. To do this, we model the RA algorithm with MSVL, specify the desired properties and then verify an instance of the RA algorithm with respect to the first-come-first-served property.

Keywords: Distributed systems, Temporal logic, Temporal logic programming, MSVL, Theorem proving

1. Introduction

With the advent of new network technologies, like Cloud Computing [AFG+10] and SOA, distributed systems have become more and more pervasive, such as peer-to-peer applications, distributed databases, network file systems and wireless sensor networks. In a distributed system, processes are at different physical locations and highly independent of each other. Further, due to the absence of a global clock, processes communicate and coordinate their actions by passing messages. Since distributed systems are inherently concurrent, asynchronous and nondeterministic, they are error-prone and tend to behave unexpectedly because of subtle bugs or race conditions. Therefore, it is a challenge to ensure the correctness of distributed systems.

Correspondence and offprint requests to: Z. Duan, E-mail: zhhduan@mail.xidian.edu.cn

Formal verification [WLBF09] is a rigorous technique for guaranteeing the formal model of distributed systems satisfies a formal specification, which mainly consists of two approaches: model checking [CGP08, DSL13] and theorem proving [Hoa78, BL84]. Model checking automatically determines whether or not a model logically satisfies a specification through exploring the model in an exhaustive way. Although it is an automatic method, it suffers from the state explosion problem. Thus, it is less suitable for data intensive applications as the treatment of data usually results in an infinite state space. Nevertheless, data intensive applications are common in distributed systems. On the contrary, theorem proving is more general and can be applied to both finite state and infinite state systems. Further, details of the proof enable users to learn more about systems.

As a competent and versatile formalism used in formal verification, temporal logic [MP92] can specify and verify concurrent systems and distributed systems. Classical temporal logics are Linear Temporal Logic (LTL) [Pnu77], Computation Temporal Logic (CTL) [CE81] and Interval Temporal Logic (ITL) [Mos86] etc. However, verification has suffered from a disadvantage that different languages have been used for writing programs, for specifying their properties and for verifying whether a program satisfies a given property. Hence, the demand for modeling, specification and verification [Jon81] in a unifying notation has led to the emergence of executable temporal logic programming languages. For example, XYZ/E [Tan83], TLA [Lam94], METATEM [BFG+90], Tempura [Mos86], MSVL [Dua06, DT08] etc. Among these languages, MSVL is a Modeling, Simulation and Verification language based on Projection Temporal Logic (PTL) [Dua96, DKH94], which is an extension of ITL. Owing to diverse temporal operators, MSVL can express many constructs, like sequence, concurrency, and non-determinacy. In order to deal with message exchanging in distributed systems, channels and asynchronous communication commands are introduced into MSVL [MWD11]. As a result, MSVL is capable of specifying distributed systems. Further, an axiomatic system of MSVL is established [YDM10], which employs Propositional PTL (PPTL) [Dua06] as the specification language. The axiomatic system is composed of two parts: state axioms and inference rules as well as interval axioms and inference rules, where the former concerns how to transform an MSVL program into its normal form within a state while the latter focuses on the deduction over an interval. At present, the axiomatic system of MSVL can be used to handle concurrent systems that communicate with shared variables. However, it cannot be applied to verify distributed systems due to the lack of mechanisms for managing message exchanging.

In order to verify distributed systems with MSVL in a deductive manner, we extend the axiomatic system of MSVL with new axioms for asynchronous communication. With the extended axiomatic system, firstly a distributed system is modeled with MSVL; secondly its desired properties are specified by PPTL; thirdly whether or not the model satisfies a property can be verified by means of the axioms and inference rules. Our contributions can be summarized as follows:

1. We define state axioms regarding asynchronous communication commands, which deduce asynchronous communication commands into their normal forms (Sect. 3). Further, we prove the soundness and completeness of the state axioms. Moreover, some useful theorems are derived for the sake of convenience.
2. In order to show how the extended axiomatic system of MSVL works for distributed systems, we apply it to the well-known Ricart–Agrawala (RA) algorithm [RA81], which is a distributed mutual exclusion algorithm and has an infinite state space (Sect. 4). To this end, we firstly model RA algorithm with MSVL and then specify the expected properties with PPTL. Further, we demonstrate a minutely verifying procedure with respect to the first-come-first-served property.

Using the extended axiomatic system of MSVL to verify distributed systems has some advantages: (1) Both MSVL and PPTL are subsets of PTL, thus modeling, specification and verification can be performed in a unified framework. Our uniform formalism elegantly avoids using two separate notations that is not convenient for users, as well as prevents time consuming and complex transformations between different formal languages in verification; (2) The specification language PPTL can express the full regular language [TD11] while both Propositional CTL (PCTL) and Propositional LTL (PLTL) fail to do so. Thus, our method can describe not only the properties like mutual exclusion, freedom of starvation, but also the interval-sensitive properties and star properties that cannot be specified by PCTL and PLTL; (3) In view of merits of theorem-proving, our proof system can be suitable for both finite and infinite systems.

This paper is organized as follows. Section 2 briefly introduces PTL, MSVL and the axiomatic system of MSVL. Further, Sect. 3 is devoted to formalizing the axioms for asynchronous communication commands. Moreover, an application to RA algorithm is supplied in Sect. 4. Finally, Sect. 5 reviews the related work and Sect. 6 draws the conclusions.

2. Preliminaries

2.1. Projection temporal logic

The underlying logic of MSVL is Projection Temporal Logic (PTL) [Dua96, Dua06, DKH94], which has two basic operators *next* and a new *projection* interpreted over infinite models.

2.1.1. Syntax

Let $Prop$ be a countable set of atomic propositions and V a countable set of variables. Both $Prop$ and V can be finite and infinite. $B \equiv \{true, false\}$ represents the boolean domain while D denotes all data domain needed by us including integers, lists, sets, etc. The terms e and formulas P of PTL are defined by the following grammar:

$$e ::= n \mid v \mid \bigcirc e \mid \boxminus e \mid f(e_1, \ldots, e_m)$$
$$P ::= p \mid e_1 = e_2 \mid F(e_1, \ldots, e_l) \mid \neg P \mid P_1 \wedge P_2 \mid \exists v : P \mid \bigcirc P \mid (P_1, \ldots, P_m)\ \mathsf{prj}\ Q \mid P^+$$

where $n \in D$ is a constant, $v \in V$, $p \in Prop$, f stands for a function, F indicates a predicate and $P, P_1, \ldots, P_m, Q$ are well-formed PTL formulas. $\neg$, $\wedge$, $\exists$ and $=$ are similar as that in classical first order logic while $\bigcirc$ (*next*), $\boxminus$ (*previous*) , $+$ (*chop-plus*) and prj (*projection*) are temporal operators. A formula (term) is called a *state formula (term)* if it does not contain any temporal operators; otherwise it is a *temporal formula (term)*. We assume variables are partitioned into static and dynamic variables. A static variable remains the same over an interval whereas a dynamic variable can have different values at different states.

2.1.2. Semantics

We define a *state* s over $V \cup Prop$ to be a pair of assignments (I_{var}, I_{prop}) where $s[v] \in I_{var}[v]$ for each variable $v \in V$ and $s[p] \in I_{prop}[p]$ for each proposition $p \in Prop$. Here $I_{var}[v]$ assigns v a value within a data domain $D' \stackrel{\text{def}}{=} D \cup \{nil\}$ with the appropriate type, in which nil means undefined, while $I_{prop}[p]$ sets p a truth value in B. An *interval* $\sigma \equiv \langle s_0, s_1, \ldots \rangle$ is a non-empty sequence of states, which can be finite or infinite. The length of σ, $|\sigma|$, is the number of states in σ minus one if σ is finite; otherwise it is ω. To have a uniform notation for both finite and infinite intervals, we will use *extended integers* as indices, that is $N_\omega = N_0 \cup \{\omega\}$, and extend the comparison operators, $=$, $<$, $\leq$, to N_ω by considering $\omega = \omega$ and for all $i \in N_0$, $i < \omega$. Moreover, we write $=$ as $\leq -\{(\omega, \omega)\}$.

To define the semantics of the projection construct, we need an auxiliary operator $\downarrow$. Let $\sigma \equiv \langle s_0, s_1, \ldots \rangle$ be an interval and $r_1, \ldots, r_h$ be integers $(h \geq 1)$ such that $0 \leq r_1 \leq \ldots \leq r_h \leq |\sigma|$. The projection of σ onto $r_1, \ldots, r_h$ is the *projected interval*, $\sigma \downarrow (r_1, \ldots, r_h) \stackrel{\text{def}}{=} \langle s_{t_1}, s_{t_2}, \ldots, s_{t_l} \rangle$, where $t_1, \ldots, t_l$ are attained from $r_1, \ldots, r_h$ by deleting all duplicates. In other words, $t_1, \ldots, t_l$ is the longest strictly increasing subsequence of $r_1, \ldots, r_h$. For instance, $\langle s_0, s_1, s_2, s_3 \rangle \downarrow (0, 2, 2, 2, 3) = \langle s_0, s_2, s_3 \rangle$. The concatenation$(\cdot)$ of an interval $\sigma \equiv \langle s_0, s_1, \ldots, s_{|\sigma|} \rangle$ with another interval $\sigma' \equiv \langle s'_0, s'_1, \ldots, s'_{|\sigma'|} \rangle$ is represented by $\sigma \cdot \sigma' \equiv \langle s_0, s_1, ..., s_{|\sigma|}, s'_0, s'_1, \ldots, s'_{|\sigma'|} \rangle$. For a variable v, we can write $\sigma' \stackrel{v}{=} \sigma$ if σ' is an interval that is the same as σ except that different values can be assigned to v.

An *interpretation* for a PTL term or formula is a tuple $I \equiv (\sigma, i, k, j)$, where $\sigma \equiv \langle s_0, s_1, \ldots \rangle$ is an interval, i and k are non-negative integers, and j is an integer or ω, such that $i \leq k \leq j \leq |\sigma|$. We write (σ, i, k, j) to mean that a term or formula is interpreted over a subinterval $\sigma_{i,\ldots,j}$ with the current state being s_k. We use I^k_{var} and I^k_{prop} to denote the state interpretation at state s_k. Each m-place function symbol f has an interpretation $I[f]$ which is a function mapping m elements in D'^m to a single value in D'. Interpretations of predicate symbols $I[F]$ are similar but map to truth values. We assume that I interprets operators such as $+$, $-$, $\cdot$, $/$ and $<$, $>$, $\leq$, $\geq$, $=$ etc standardly. The evaluation of e relative to $I \equiv (\sigma, i, k, j)$ is defined as $I[e]$ shown in Table 1 while the satisfaction relation $\models$ for formulas is given in Table 2.

A formula P is satisfied by an interval σ, signified by $\sigma \models P$ if $(\sigma, 0, 0, |\sigma|) \models P$. A formula P is called *satisfiable* if $\sigma \models P$ for some σ. Further, P is said to be *valid*, denoted by $\models P$, if $\sigma \models P$ for all intervals σ.

Table 1. Interpretation of PTL terms [Dua06]

$I[n]$	$\triangleq$	$n \triangleq D$
$I[v]$	$\triangleq$	$s_k[v] \triangleq I^k_{var}[v]$
$I[f(e_1, \ldots, e_m)]$	$\triangleq$	$\begin{cases} I[f](I[e_1], \ldots, I[e_m]) & \text{if } I[e_h] \triangleq nil \text{ for all } 1 \le h \le m \\ nil & \text{otherwise} \end{cases}$
$I[\bigcirc e]$	$\triangleq$	$\begin{cases} (\sigma, i, k+1, j)[e] & \text{if } k < j \\ nil & \text{otherwise} \end{cases}$
$I[\boxminus e]$	$\triangleq$	$\begin{cases} (\sigma, i, k-1, j)[e] & \text{if } i < k \\ nil & \text{otherwise} \end{cases}$

Table 2. Interpretation of PTL formulas [Dua06]

$I \models p$	iff $s_k[p] \triangleq I^k_{prop}[p] \triangleq true$
$I \models F(e_1, \ldots, e_l)$	iff $I[F](I[e_1], \ldots, I[e_l]) \triangleq true$, and $I[e_h] \triangleq nil$ for all $1 \le h \le l$
$I \models e_1 \triangleq e_2$	iff $I[e_1] \triangleq I[e_2]$
$I \models \neg P$	iff $I \not\models P$
$I \models P \triangleq Q$	iff $I \models P$ and $I \models Q$
$I \models \bigcirc P$	iff $k < j$ and $(\sigma, i, k+1, j) \models P$
$I \models \forall v : P$	iff $(\sigma', i, k, j) \models P$ for some $\sigma' \overset{v}{=} \sigma$
$I \models (P_1, \ldots, P_m)$ prj Q	iff there exist integers $r_0, \ldots, r_m$ and $k \triangleq r_0 \le \ldots \le r_{m-1} \triangleq r_m \le j$ such that $(\sigma, i, r_0, r_1) \models P_1$ and $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P_l$ for all $1 < l \le m$ and $(\sigma', 0, 0, \lvert\sigma'\rvert) \models Q$ for σ' given by : (1) $r_m < j$ and $\sigma' \triangleq \sigma \downarrow (r_0, \ldots, r_m) \cdot \sigma_{(r_m+1, \ldots, j)}$ (2) $r_m \triangleq j$ and $\sigma' \triangleq \sigma \downarrow (r_0, \ldots, r_h)$ for some $0 \le h \le m$
$I \models P^+$	iff there are finitely many integers $r_0, \ldots, r_n$ and $k \triangleq r_0 \le r_1 \le \cdots \le r_{n-1} \triangleq r_n \triangleq j$ $(n \ge 1)$ such that $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P$ for all $1 \le l \le n$ or $j \triangleq \omega$ and there are infinitely many integers $k \triangleq r_0 \le r_1 \le r_2 \le \ldots$ such that $\lim_{i \to \infty} r_i \triangleq \omega$ and for all $l \ge 1$, $(\sigma, r_{l-1}, r_{l-1}, r_l) \models P$

2.1.3. Derived formulas

Some derived formulas from elementary PTL formulas are formalized below for the sake of convenience. The abbreviations true, false, $\triangle$, $\rightarrow$ and $\leftrightarrow$ are defined as usual. 'ε' signifies the final state; 'more' represents that the current state is a non-final state of an interval; 'len(n)' specifies the distance is n from the current state to the final state of an interval; 'skip' denotes that the length of the interval is one; '$P^{\triangle}$' (*chop-star*) means that P is not executed or is repeatedly executed a finite or infinite number of times; '$\Diamond P$' (*sometimes*) states that P will hold eventually in the future including the current state; '$\Box P$' (*always*) declares that P always holds in the future from now on; 'halt(P)' is true over an interval if and only if P is true at the final state; 'fin(P)' is true as long as P is true at the final state; 'keep(P)' is true if P is true at each state ignoring the final one. Sometimes, $\models \Box(P \leftrightarrow Q)$ is represented by $P \equiv Q$ (*strong equivalent*), meaning that P and Q have the same truth at all states in every model.

ε	$\overset{def}{=}$	$\neg \bigcirc$ true	len(n)	$\overset{def}{=}$	$\begin{cases} \varepsilon & \text{if } n \triangleq 0 \\ \bigcirc \text{len}(n-1) & \text{if } n > 1 \end{cases}$
$P^{\triangle}$	$\overset{def}{=}$	$P^+ \triangle \varepsilon$	$\Diamond P$	$\overset{def}{=}$	(true, P) prj ε
more	$\overset{def}{=}$	$\neg \varepsilon$	halt(P)	$\overset{def}{=}$	$\Box(\varepsilon \leftrightarrow P)$
skip	$\overset{def}{=}$	len(1)	$\Box P$	$\overset{def}{=}$	$\neg \Diamond \neg P$
fin(P)	$\overset{def}{=}$	$\Box(\varepsilon \rightarrow P)$	keep(P)	$\overset{def}{=}$	$\Box(\neg\varepsilon \rightarrow P)$

Propositional PTL (PPTL) is a propositional subset of PTL, whose syntax is below:

$$P ::\triangle\ p \mid \neg P \mid P_1 \triangle P_2 \mid \bigcirc P \mid (P_1, \ldots, P_m) \text{ prj } Q \mid P^+$$

Table 3. MSVL programs [Dua06]

Termination:	ε
Positive immediate assignment:	$x \Box e \stackrel{\text{def}}{=} x \Box e \Box p_x$
Unification:	$x \Box e$
Assignment:	$x :\Box e \stackrel{\text{def}}{=} \Box x \Box e \Box \mathsf{skip}$
Projection:	$(P_1, \ldots, P_m)\ \mathsf{prj}\ Q$
Sequential statement:	$P\ ;\ Q \stackrel{\text{def}}{=} (P, Q)\ \mathsf{prj}\ \varepsilon$
Conjuction:	$P \Box Q$
State frame:	$\mathsf{lbf}(x) \stackrel{\text{def}}{=} \neg\mathsf{af}(x) \rightarrow \Box\ a : (\boxminus x \Box a \Box x \Box a)$
Selection:	$P \Box Q$
Interval frame:	$\mathsf{frame}(x) \stackrel{\text{def}}{=} \Box(\mathsf{more} \rightarrow \Box\ \mathsf{lbf}(x))$
Next statement:	$\bigcirc P$
Conditional statement:	$\mathtt{if}\ b\ \mathtt{then}\ P\ \mathtt{else}\ Q \stackrel{\text{def}}{=} (b \Box P) \Box (\neg b \Box Q)$
Always statement:	$\Box P$
While statement:	$\mathtt{while}\ b\ \mathtt{do}\ P \stackrel{\text{def}}{=} (P \Box b)^{\Box} \Box \Box(\varepsilon \rightarrow \neg b)$
Existential quantification:	$\exists x : P(x)$
Parallel:	$P \Box Q \stackrel{\text{def}}{=} (P \Box (Q\ ;\mathsf{true})) \Box (Q \Box (P\ ;\mathsf{true}))$
Synchronized communication:	$\mathsf{await}(b) \stackrel{\text{def}}{=} \mathsf{halt}(b) \Box \mathsf{frame}(V_b)$

2.2. MSVL

MSVL [Dua06, DT08] is a Modeling, Simulation and Verification language, which provides an executable PTL framework with more succinct description and immediate practical application. In MSVL, expressions can be regarded as PTL terms while statements can be considered as PTL formulas. The arithmetic expression e and boolean expression b of MSVL are inductively defined as follows, where n is a constant, x a variable:

$$e ::\Box\ n \mid x \mid \Box x \mid \boxminus x \mid e_0\ op\ e_1 \qquad (op ::\Box\ + \mid - \mid \Box \mid /)$$
$$b ::\Box\ \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_0 \Box b_1 \mid e_0 \Box e_1 \mid e_0 < e_1$$

A program in MSVL can be formalized in Table 3. 'ε' is the termination statement, which simply states that the current state is the final state of the interval over which a program is executed. The next statement '$\bigcirc P$' means that P holds at the immediate successive state. '$\Box P$' implies that P is always true in all states from now on until the termination of an interval. The sequential statement '$P\ ;\ Q$' signifies a computation of P followed by Q, that is, P keeps executing from the current state until some point in future at which it terminates and Q will start executing. The conditional statement '$\mathtt{if}\ b\ \mathtt{then}\ P\ \mathtt{else}\ Q$' and while statement '$\mathtt{while}\ b\ \mathtt{do}\ P$' can be illustrated as that in the conventional imperative languages. '$\mathtt{if}\ b\ \mathtt{then}\ P\ \mathtt{else}\ Q$' first evaluates the boolean expression; if b is *true*, the process P is executed; otherwise Q is executed. The iteration '$\mathtt{while}\ b\ \mathtt{do}\ P$' allows process P to be repeatedly executed a finite (or an infinite) number of times over a finite (resp. an infinite) interval as long as the condition b is satisfied at the beginning of each execution. If b becomes *false*, the `while` statement terminates.

In the unification statement '$x \Box e$', if e is evaluated to a constant in D and x has not been specified at the current state or was specified to the same value as e, we say x is unified with e. In this case, the equality $x \Box e$ is satisfied; otherwise it is false. The assignment statement '$x :\Box e$' claims that at the current state e is first evaluated to a constant and then at the next state x equals the constant, which is executed over the interval with the length 1. '$P \Box Q$' represents the selection statement asserting that P or Q is executed with non-determinacy. The existential quantification statement '$\exists x : P(x)$' intends to hide the variable x within the process P and may allow a process P to take advantage of a local variable.

The conjunction statement '$P \Box Q$' declares that the processes P and Q are executed concurrently sharing all the states during the mutual execution. The parallel construction '$P \Box Q$' shows another concurrent computation manner. The distinguished difference between '$P \Box Q$' and '$P \Box Q$' is that the former permits both P and Q to be autonomous, or rather to specify their own intervals while the latter does not. E.g., $\mathsf{len}(2) \Box \mathsf{len}(3)$ holds but $\mathsf{len}(2) \Box \mathsf{len}(3)$ is obviously false. Projection can be treated as a special parallel computation with greater autonomy, which is executed on two different time scales. '$(P_1, \ldots, P_m)\ \mathsf{prj}\ Q$' tells us that Q is executed in parallel with $P_1, \ldots, P_m$ over an interval obtained by taking the endpoints of the intervals over which the $P_i^{\Box}$s are executed. In this construct, processes $P_1, \ldots, P_m, Q$ are self-governing and each of them has the right to specify the interval over which it is executed. In particular, the sequence of $P_i^{\Box}$s and Q may terminate at different points.

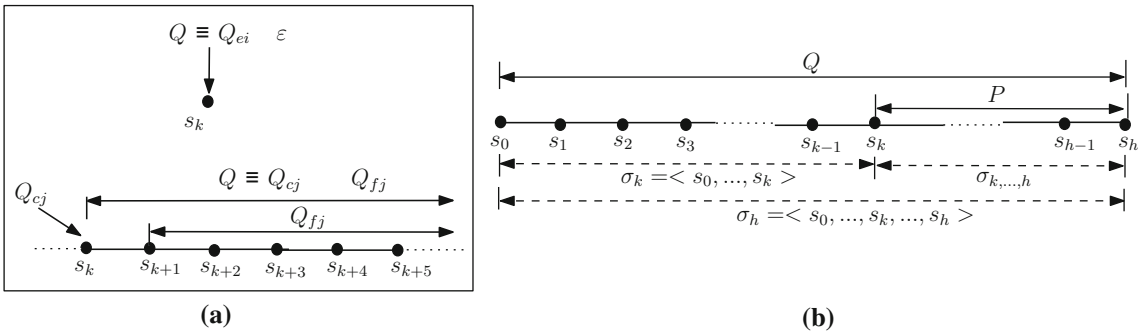

Fig. 1. Intuitive meaning of the normal form and the triple $\{\sigma_k, A\}\ P\ \{\sigma_h, B\}$

Framing concerns how the value of a variable can be carried from one state to the next. The crux is how to perceive the assignments of values to variables. To identify an occurrence of an assignment to a variable, say x, we make use of a flag called the *assignment flag*, denoted by predicate $\mathsf{af}(x)$; it is *true* whenever an assignment of a value to x is encountered, and *false* otherwise. To define $\mathsf{af}(x)$, a new assignment $x \Box e \stackrel{\text{def}}{=} x \Box e \Box p_x$ is needed, where p_x is an atomic proposition connected with the variable x and cannot be used for other purposes. Then the assignment flag is formalized as $\mathsf{af}(x) \stackrel{\text{def}}{=} p_x$. As expected, when $x \Box e$ is encountered, p_x is set to *true*, hence $\mathsf{af}(x)$ is *true*; if no assignment to x takes place, p_x is unspecified. In this case, we will use the minimal model [Dua06] to force it to be *false*. Note that the definition given above is one way to specify $\mathsf{af}(x)$ and there may be some other methods to formulate it. Armed with $\mathsf{af}(x)$ we can define state frame $\mathsf{lbf}(x)$ and interval frame $\mathsf{frame}(x)$. Intuitively, '$\mathsf{lbf}(x)$' means that, when a variable is framed at a state, its value remains unchanged if no assignment is encountered at that state while '$\mathsf{frame}(x)$' states that a variable is framed over an interval if it is framed at each state over the interval. Introducing the framing operator enables us to define the synchronizing construct $\mathsf{await}(b)$, where V_b represents all dynamic variables contained in b. The await statement is employed to synchronize between parallel processes with shared variables. It does not change any variables, but waits until the condition b becomes true, at which point it terminates.

Further, we have the following useful logic laws, where w is a state formula. Their proofs can be found in [Dua06].

(L1) $\varepsilon ; P \equiv P$ (L2) $w \Box (P ; Q) \equiv (w \Box P) ; Q$
(L3) $\Box P ; Q \equiv \Box (P ; Q)$ (L4) $(P_1 \Box P_2) ; Q \equiv (P_1 ; Q) \Box (P_2 ; Q)$
(L5) $\mathsf{await}(b) \equiv b \Box \varepsilon \Box \neg b \Box \Box (\mathsf{lbf}(V_b) \Box \mathsf{await}(b))$ (L6) $P \equiv Q \Box\Box\ prog[P] \equiv prog[Q/P]$

Definition 2.1 (*Normal Form, NF*) Let Q be a program in MSVL. The normal form of Q is defined as

$$Q \stackrel{\text{def}}{=} \bigvee_{i\Box 1}^{k} (Q_{ei} \Box \varepsilon) \Box \bigvee_{j\Box 1}^{h} (Q_{cj} \Box \Box Q_{fj})$$

where $k + h \geq 1$ and the following hold:

- Q_{fj} is a general program;
- each Q_{ei} and Q_{cj} is either true or a state formula of the form: $P_1 \Box \cdots \Box P_m$ $(m \geq 1)$ such that each P_l $(1 \leq l \leq m)$ is either $x \Box n$ with $x \Box V$, $n \Box D$, or $\dot{p}_{x_l}$ denoting p_{x_l} or $\neg p_{x_l}$.

In some cases, we simply write $Q_e \Box \varepsilon$ instead of $\bigvee_{i\Box 1}^{k} (Q_{ei} \Box \varepsilon)$. If Q terminates at the current state it is reduced to $Q_e \Box \varepsilon$; otherwise it is reduced to $Q_{cj} \Box \Box Q_{fj}$. The conjuncts $Q_e \Box \varepsilon$ and $Q_{cj} \Box \Box Q_{fj}$ are named *basic terminal products* and *basic future products* respectively. Moreover, Q_{cj} (or Q_e) and $\Box Q_{fj}$ are called *present components* and *future components* respectively. The intuitive meaning is shown in Fig. 1a.

Table 4. State axioms and state inference rules [YDM10]

(A1)	$\mathsf{lbf}(x) \Box x \Box n \Box\!\!\!\equiv x \Box n \Box p_x(n \Box \boxminus x)$	(A2)	$\mathsf{lbf}(x) \Box x \Box n \Box\!\!\!\equiv x \Box n \Box p_x$
(A3)	$\Box P \Box \mathsf{more} \Box\!\!\!\equiv P \Box \Box\Box P$	(A4)	$\Box P \Box \varepsilon \Box\!\!\!\equiv P \Box \varepsilon$
(A5)	$\mathsf{frame}(x) \Box \varepsilon \Box\!\!\!\equiv \varepsilon$	(A6)	$\mathsf{frame}(x) \Box \mathsf{more} \Box\!\!\!\equiv \Box(\mathsf{lbf}(x) \Box \mathsf{frame}(x))$
(A7)	$\Box P ; Q \Box\!\!\!\equiv \Box(P ; Q)$	(A8)	$\varepsilon ; Q \Box\!\!\!\equiv Q$
(A9)	$(w \Box P) ; Q \Box\!\!\!\equiv w \Box (P ; Q)$	(A10)	$P \parallel Q \Box\!\!\!\equiv ((P ; \mathsf{true}) \Box Q) \Box ((Q ; \mathsf{true}) \Box P)$
(A11)	$\Box P \Box Q \Box\!\!\!\equiv \Box P \Box Q \Box \mathsf{more}$	(A12)	$\mathtt{if}\ b\ \mathtt{then}\ P\ \mathtt{else}\ Q \Box\!\!\!\equiv (b \Box P) \Box (\neg b \Box Q)$
(A13)	$\varepsilon\ \mathsf{prj}\ Q \Box\!\!\!\equiv Q$	(A14)	$(P_1, \ldots, P_m)\ \mathsf{prj}\ \varepsilon \Box\!\!\!\equiv P_1 ; \ldots ; P_m$
(A15)	$(P_1, \ldots, P_{i-1}, w \Box \varepsilon, P_{i+1}, \ldots, P_m)\ \mathsf{prj}\ Q \Box\!\!\!\equiv (P_1, \ldots, P_{i-1}, w \Box P_{i+1}, \ldots, P_m)\ \mathsf{prj}\ Q$		
(A16)	$(w \Box P_1, P_2, \ldots, P_m)\ \mathsf{prj}\ Q \Box\!\!\!\equiv w \Box (P_1, \ldots, P_m)\ \mathsf{prj}\ Q$		
(A17)	$(P_1, \ldots, P_m)\ \mathsf{prj}\ (w \Box Q) \Box\!\!\!\equiv w \Box (P_1, \ldots, P_m)\ \mathsf{prj}\ Q$		
(A18)	$(\Box P_1, \ldots, P_m)\ \mathsf{prj}\ \Box Q \Box\!\!\!\equiv \Box(P_1 ; (P_2, \ldots, P_m)\ \mathsf{prj}\ Q)$		
(A19)	$\mathtt{while}\ b\ \mathtt{do}\ P \Box\!\!\!\equiv \mathtt{if}\ b\ \mathtt{then}\ (P \Box \mathsf{more} ; \mathtt{while}\ b\ \mathtt{do}\ P)\ \mathtt{else}\ \varepsilon$		
(A20)	$\mathsf{await}(b) \Box\!\!\!\equiv b \Box \varepsilon \Box \neg b \Box \Box(\mathsf{lbf}(V_b) \Box \mathsf{await}(b))$		
(A21)	$x :\Box n \Box\!\!\!\equiv \Box(x \Box n \Box \varepsilon)$		
(A22)	$\Box P$, where P is a substitution instance of all valid formulas.		
(R1)	$P \Box\!\!\!\equiv Q \Box\Box\ prog[P] \Box\!\!\!\equiv prog[Q/P]$		
(R2)	$P(x) \Box\!\!\!\equiv (P_e(x) \Box \varepsilon) \Box (P_c(x) \Box \Box P_f(x)) \Box\Box \Box x : P(x) \Box\!\!\!\equiv (\Box x : P_e(x) \Box \varepsilon) \Box (\Box x : P_c(x) \Box \Box \Box x : P_f(x))$		
(R3)	$\Box P \Box\Box \Box \Box P$, where P is a substitution instance of all valid formulas.		

2.3. Axiomatic semantics of MSVL

The axiomatic semantics of MSVL was given in [YDM10] and here we just briefly present the framework. For convenience, we represent $\Box \Box(P \leftrightarrow Q)$ by $P \Box\!\!\!\equiv Q$. In general, the axiomatic semantics can be classified into two categories: one for state deduction and the other for interval deduction. Thereinto, the state deduction concerns how to transform an MSVL program into its normal form within a state while the interval deduction is regarding programs deduced over an interval and simultaneously properties verified over the interval.

A set of state axioms and state inference rules for the state deduction is shown in Table 4, where n is a constant, w is a state formula, and $P_e(x)$ and $P_c(x)$ are present components. State axiom (A1) claims that when n is different from the previous value of x, $x \Box n$ in conjunction with $\mathsf{lbf}(x)$ can be deduced to $x \Box n$. It is similar for axiom (A2). In the axiomatic semantics, $\mathsf{lbf}(x)$ can be regarded as a state formula. Axioms (A3) and (A4) care for always statement whereas axioms (A5) and (A6) for interval frame statement. They are given in terms of the length of an interval: zero (i.e ε) and non-zero (i.e. more). Further, axioms (A7–A9) are specified for sequential statement $P ; Q$. Moreover, axioms (A10), (A12) and (A19–A21) respectively handle parallel, conditional, while, await and assignment statements by means of their own definitions. Axiom (A11) deals with $\Box P$. Axioms (A13–A18) are concerned with projection statement $(P_1, \ldots, P_m)\ \mathsf{prj}\ Q$. Furthermore, axiom (A22) describes the substitution instances of all valid classical first order formulas within temporal contexts and some algebraic properties of temporal operators. For example, temporal operators ($\Box$) and (;) have the *distributive* property over $\Box$ and $\Box$, that is, $(P \Box Q) ; R \Box\!\!\!\equiv (P ; R) \Box (Q ; R)$; $\Box P \Box \Box Q \Box\!\!\!\equiv \Box(P \Box Q)$.

State inference rule (R1) is for equivalent replacement, which tells us that provided that a program $prog[P]$ involves a subprogram P and $P \Box\!\!\!\equiv Q$, we can substitute some occurrences of P in $prog$ by Q and obtain an equivalent program $prog[Q/P]$. Rule (R2) asserts that $\Box x : P(x)$ can be deduced to its normal form as long as $P(x)$ has been in a normal form. Further, rule (R3) declares that if $\Box P$, then P always holds over intervals, i.e $\Box \Box P$.

Theorem 2.1 *Some results for the state axioms and inference rules* [YDM10]*:*

1. *Any MSVL program P can be deduced into normal form by state axioms and inference rules in Table 4.*
2. *Let P and Q be MSVL programs. State axioms and inference rules in Table 4 are sound and complete, that is $P \Box\!\!\!\equiv Q \Box\Box\ P \equiv Q$ and $P \equiv Q \Box\Box\ P \Box\!\!\!\equiv Q$.*

In the axiomatic semantics, PPTL [Dua06] is employed as the assertion language, whose NF is similar as that in Definition 2.1 and formalized in [Dua06]. Let A, B be PPTL formulas and P be an MSVL program. Thus, the correctness assertion can be defined in the form of a variation of Hoare's triple as

$$\{\sigma_k, A\}\ P\ \{\sigma_h, B\}$$

Table 5. Axioms over intervals [YDM10]

(AEM)	$\{\sigma_k, \varepsilon\}\ \varepsilon\ \{\sigma_k, \varepsilon\}$ (APC) $\{\sigma_k, A_c\}\ P_c\ \{\sigma_k, A_c\}$ if $P_c \rightarrow A_c$
(ISR)	$\{\sigma_k, A\}\ prog[P]\ \{\sigma_h, B\}$ and $P \boxminus Q$ $\Box\Box$ $\{\sigma_k, A\}\ prog[Q/P]\ \{\sigma_h, B\}$ and $P \boxminus Q$
(AAS)	$\{\sigma_k, A\}\ x := m \Box sp(y) \Box P\ \{\sigma_h, B\}$ $\Box\Box$ $\{\sigma_k, A\}\ x := m \Box sp(y)[m/x] \Box P\ \{\sigma_h, B\}$ where $\mathsf{lbf}(y)$ and $\mathsf{lbf}(x)$ does not occur in P.
(LBF)	$\{\sigma_k, A\}\ \mathsf{lbf}(x) \Box P\ \{\sigma_h, B\}$ $\Box\Box$ $\{\sigma_k, A\}\ x := \boxminus x \Box P\ \{\sigma_h, B\}\ (k \geq 1)$ where $x \Box c$ and $x \Box c$ does not occur in P.
(ANext)	$\{\sigma_k, A\}\ P_c \Box \Box P_f\ \{\sigma_h, B\}$ $\Box\Box$ $\begin{cases} \{\sigma_k[P_c] \cdot \Box s_{k+1}\Box, A_f\}\ P_f\ \{\sigma_h, B\} \text{ and } \{\sigma_k, A_c\}\ P_c\ \{\sigma_k, A_c\} & \text{if } P_c \rightarrow A_c \\ \{\sigma_k, \text{false}\}\ P_c \Box \Box P_f\ \{\sigma_h, B\} & \text{otherwise} \end{cases}$ where $A \equiv (A_c \Box \Box A_f) \Box (A_e \Box \varepsilon)$
(AEmpty)	$\{\sigma_k, A\}\ P_e \Box \varepsilon\ \{\sigma_h, B\}$ $\Box\Box$ $\begin{cases} \{\sigma_k[P_e], \varepsilon\}\ \varepsilon\ \{\sigma_k[P_e], \varepsilon\} \text{ and } \{\sigma_k, A_e \Box B\}\ P_e \{\sigma_k, A_e \Box B\} & \text{if } P_e \rightarrow A_e \Box B \\ \{\sigma_k, \text{false}\}\ P_e \Box \varepsilon\ \{\sigma_k, \text{false}\} & \text{otherwise} \end{cases}$ where $A \equiv (A_c \Box \Box A_f) \Box (A_e \Box \varepsilon)$
(SSR)	$\{\sigma_k, A\}\ P_i\ \{\sigma_h, B\}$ $\Box\Box$ $\{\sigma_k, A\}\ P_1 \Box P_2\ \{\sigma_h, B\}\ (i \Box 1, 2)$
(EQR1)	$\{\sigma_k, A\}\ P_c(y) \Box \Box P\ \{\sigma_h, B\}$ $\Box\Box$ $\{\sigma_k, A\}\ \Box x : P_c(x) \Box \Box P\ \{\sigma_h, B\}$
(EQR2)	$\{\sigma_k, A\}\ P_e(y) \Box \varepsilon\ \{\sigma_h, B\}$ $\Box\Box$ $\{\sigma_k, A\}\ \Box x : P_e(x) \Box \varepsilon\ \{\sigma_h, B\}$

Thereinto, A and $\sigma_k \Box \Box s_0, \ldots, s_k\Box$ are *left-conditions*, where σ_k represents an interval over which some program Q has been deduced for k states $s_0, \ldots, s_{k-1}$ $(k \geq 1)$ and implies that the successor of Q, program P, is being deduced at state s_k; B and $\sigma_h \Box \Box s_0, \ldots, s_h\Box$ are *right-conditions*, where σ_h indicates the whole interval over which the original program Q will be deduced for $h + 1$ states $s_0, \ldots, s_h$ $(h \geq 0)$ and over $\Box s_k, \ldots, s_h\Box$ program P will be deduced. Actually, σ_k is a prefix of σ_h. The intuitive interpretation of the triple is displayed in Fig. 1b. During the deduction, if $\sigma_{(k\ldots h)}$ satisfies program P, then it should satisfy condition A and the final state s_h should satisfy condition B. If P terminates, then σ_h is a finite interval; otherwise, it is an infinite interval, denoted by σ_ω $(h \Box \omega)$. We often call $\{\sigma_k, A\}\ P\ \{\sigma_h, B\}$ a *general triple*. When $k \Box 0$, $\{\sigma_0, A\}\ P\ \{\sigma_h, B\}$ is an *initial triple*, abbreviated as $\{A\}\ P\ \{B\}$, which means that program P is deduced over interval σ_h starting from the initial state s_0.

In addition, a group of interval axioms and inference rules is presented in Table 5, where P_c, P_e, A_c and A_e are present components; $sp(x)$ is $x \Box e$, $x \Box e$ or a boolean expression. Axioms (AEM) and (APC) bear on the termination statement and the present components respectively. Rule (ISR) is the counterpart over an interval of the state inference rule (R1). Rule (AAS) focuses on evaluating $sp(y)$ via *substitution* while rule (LBF) is concerned with state frame $\mathsf{lbf}(x)$. Rule (ANext) copes with future products and deduces a program $P_c \Box \Box P_f$ from the current state s_k to the next state s_{k+1} with present components P_c deduced at s_k and '*next*' program P_f deduced over a sub-interval starting from state s_{k+1}. Further, rule (AEmpty) handles terminal products and terminates the verification at the state s_k. Rule (SSR) is for selection statement $P_1 \Box P_2$ while rules (EQR1) and (EQR2) for existential quantification statement $\Box x : P(x)$.

Thus, an axiomatic system of MSVL is established. For the correctness assertion $\{\sigma_0, A\}\ P\ \{\sigma_h, B\}$, the deduction proceeds as follows: at first, the program P is deduced into its normal form by the state axioms and inference rules and A is reduced into its normal form by the logical laws in [Dua06]; further, if one of the present components of P satisfies one of the present components of A, the deduction is transferred to the next state with the interval rules; the process is repeated over and over in a similar fashion until the termination of an interval. The axiomatic system is proved to be sound and relatively complete [YDM10], that is for any correctness assertion $\{A\}\ P\ \{B\}$, $\Box\ \{A\}\ P\ \{B\}$ $\Box\Box$ $\models \Box\ \{A\}\ P\ \{B\}$ and $\models \Box\ \{A\}\ P\ \{B\}$ $\Box\Box$ $\vdash \Box\ \{A\}\ P\ \{B\}$.

3. Asynchronous communication axioms

Generally, distributed systems communicate asynchronously by means of message passing. Nevertheless, the axiomatic system of MSVL in Sect. 2.3 lacks mechanisms to deal with asynchronous communication. Thus, in this section, we enrich the axiomatic system with some new axioms to make it suitable for distributed systems.

Table 6. Asynchronous communication axioms

(A23)	$\texttt{send}(c, x) \Rightarrow c :\Box\ c \cdot \Box x\Box$	if *isfull*(c) is false
(A24)	$\texttt{send}(c, x) \Rightarrow \Box(\mathsf{lbf}(c) \Box \texttt{send}(c, x))$	if *isfull*(c) is true
(A25)	$\texttt{receive}(c, x) \Rightarrow x :\Box\ head(c) \Box\ c :\Box\ tail(c)$	if *isempty*(c) is false
(A26)	$\texttt{receive}(c, x) \Rightarrow \Box(\mathsf{lbf}(c) \Box \texttt{receive}(c, x))$	if *isempty*(c) is true
(A27)	$\texttt{put}(c, x) \Rightarrow c :\Box\ c \cdot \Box x\Box$	if *isfull*(c) is false
(A28)	$\texttt{put}(c, x) \Rightarrow \mathsf{skip}$	if *isfull*(c) is true
(A29)	$\texttt{get}(c, x) \Rightarrow x :\Box\ head(c) \Box\ c :\Box\ tail(c)$	if *isempty*(c) is false
(A30)	$\texttt{get}(c, x) \Rightarrow \mathsf{skip}$	if *isempty*(c) is true

3.1. Asynchronous communication commands

To specify message exchanging in distributed systems, [MWD11] introduced asynchronous communication commands into MSVL. We here only present the main idea. In MSVL, a channel is defined as a first-in-first-out (FIFO) list. For instance, $\Box 1, 2\Box$ is a channel containing two elements and $\Box\Box$ is a null channel. Further, based on channels, two couples of asynchronous communication commands are formalized below:

$$
\begin{array}{lll}
\texttt{send}(c, x) & \stackrel{\text{def}}{=} & \mathsf{await}(\neg \mathit{isfull}(c))\ ;\ c :\Box\ c \cdot \Box x\Box \\
\texttt{receive}(c, x) & \stackrel{\text{def}}{=} & \mathsf{await}(\neg \mathit{isempty}(c))\ ;\ x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c) \\
\texttt{put}(c, x) & \stackrel{\text{def}}{=} & \texttt{if}(\neg \mathit{isfull}(c))\ \texttt{then}\{c :\Box\ c \cdot \Box x\Box\}\ \texttt{else}\ \{\mathsf{skip}\} \\
\texttt{get}(c, x) & \stackrel{\text{def}}{=} & \texttt{if}(\neg \mathit{isempty}(c))\ \texttt{then}\{x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c)\}\ \texttt{else}\ \{\mathsf{skip}\}
\end{array}
$$

where c is a channel and x is a variable; *isfull*(c) and *isempty*(c) are predicates to check the status of the channel c and are respectively true when c is full and empty; *head* and *tail* are functions to respectively fetch the first element and the tail list of c.

Commands '`send`' and '`put`' are for sending messages while commands '`receive`' and '`get`' for receiving messages. Usually, '`send`' and '`receive`' are used together for communication without timeout mechanism whereas '`put`' and '`get`' are employed for communication with timing restrictions. The command '$\texttt{send}(c, x)$' inserts the message x into the tail of channel c when c is not full. If c is full, the statement will wait until c has an empty place for x. '$\texttt{receive}(c, x)$' removes the first message from c and stores it into x when c is not empty. If c is empty, it will wait until some messages appear in c. Further, '$\texttt{put}(c, x)$' firstly determines whether or not the channel c is full; if this is true, it terminates without doing anything; otherwise, it puts x at the end of c. Moreover, at first, '$\texttt{get}(c, x)$' decides whether or not the channel c is empty; if it is empty, it does nothing; or else, it acquires the head message from c and stores it into x.

3.2. State axioms for asynchronous communication

Some state axioms for asynchronous communication commands are defined in Table 6, which are on the basis of their definitions and discussed in terms of the status of a channel c.

- Axioms (A23–24) are concerned with '$\texttt{send}(c, x)$'. Axiom (A23) deals with the case when the channel c is not full and put the message x at the tail of c while (A24) treats with the situation when c is full and deduces '$\texttt{send}(c, x)$' from the current state to the next state.
- Axioms (A25–26) are for '$\texttt{receive}(c, x)$'. Axiom (A25) considers the deduction of '$\texttt{receive}(c, x)$' when c is not empty whereas (A26) bears on that when c is empty.
- Axioms (A27–28) focus on '$\texttt{put}(c, x)$'. Axiom (A27) deduces '$\texttt{put}(c, x)$' into $c :\Box\ c \cdot \Box x\Box$ on the condition that c is not full while (A28) deduces '$\texttt{put}(c, x)$' into skip when c is full.
- Axioms (A29–30) care for '$\texttt{get}(c, x)$'. Axiom (A29) handles the situation that c is not empty whereas (A30) copes with the circumstance that c is empty.

As a matter of fact, with asynchronous communication axioms in Table 6 and state axioms and inference rules in Table 4, we can deduce asynchronous communication commands into their normal forms, which is declared by Theorem 3.1.

Theorem 3.1 *Statements* $\mathtt{send}(c, x)$, $\mathtt{receive}(c, x)$, $\mathtt{put}(c, x)$ *and* $\mathtt{get}(c, x)$ *can be deduced into normal form by state axioms and state inference rules in Tables 4 and 6.*

Proof. We prove them one by one and rule (R1) is used impliedly:

(1) $\mathtt{send}(c, x)$:

$\mathtt{send}(c, x)$
$= \mathtt{send}(c, x) \Box \mathsf{true}$ (A22, R3)
$= \mathtt{send}(c, x) \Box (isfull(c) \Box \neg isfull(c))$ (A22, R3)
$= \mathtt{send}(c, x) \Box isfull(c) \Box \mathtt{send}(c, x) \Box \neg isfull(c)$ (A22, R3)
$= isfull(c) \Box \Box(\mathsf{lbf}(c) \Box \mathtt{send}(c, x)) \Box \neg isfull(c) \Box c :\Box c \cdot \Box x\Box$ (A23, A24)
$= isfull(c) \Box \Box(\mathsf{lbf}(c) \Box \mathtt{send}(c, x)) \Box \neg isfull(c) \Box \Box(c \Box c^{\Box} \Box \varepsilon)$ (A21)

where $c^{\Box} \Box c \cdot \Box x\Box$ can be considered as a constant. If the predicate '$isfull(c)$' is true, the normal form of $\mathtt{send}(c, x)$ is $\Box(\mathsf{lbf}(c) \Box \mathtt{send}(c, x))$; otherwise, its normal form is $\Box(c \Box c^{\Box} \Box \varepsilon)$.

(2) $\mathtt{receive}(c, x)$:

$\mathtt{receive}(c, x)$
$= \mathtt{receive}(c, x) \Box \mathsf{true}$ (A22, R3)
$= \mathtt{receive}(c, x) \Box (isempty(c) \Box \neg isempty(c))$ (A22, R3)
$= \mathtt{receive}(c, x) \Box isempty(c) \Box \mathtt{receive}(c, x) \Box \neg isempty(c)$ (A22, R3)
$= isempty(c) \Box \Box(\mathsf{lbf}(c) \Box \mathtt{receive}(c, x)) \Box \neg isempty(c) \Box x :\Box head(c) \Box c :\Box tail(c)$ (A25, A26)
$= isempty(c) \Box \Box(\mathsf{lbf}(c) \Box \mathtt{receive}(c, x)) \Box \neg isempty(c) \Box \Box(x \Box head(c) \Box c \Box c^{\Box} \Box \varepsilon)$ (A21)

where $c^{\Box} \Box tail(c)$ is a constant. If the predicate '$isempty(c)$' is true, the normal form of $\mathtt{receive}(c, x)$ is $\Box(\mathsf{lbf}(c) \Box \mathtt{receive}(c, x))$; or else, its normal form is $\Box(x \Box head(c) \Box c \Box c^{\Box} \Box \varepsilon)$.

(3) $\mathtt{put}(c, x)$:

$\mathtt{put}(c, x)$
$= \mathtt{put}(c, x) \Box \mathsf{true}$ (A22, R3)
$= \mathtt{put}(c, x) \Box (isfull(c) \Box \neg isfull(c))$ (A22, R3)
$= \mathtt{put}(c, x) \Box isfull(c) \Box \mathtt{put}(c, x) \Box \neg isfull(c)$ (A22, R3)
$= isfull(c) \Box \mathsf{skip} \Box \neg isfull(c) \Box c :\Box c \cdot \Box x\Box$ (A27, A28)
$= isfull(c) \Box \Box \varepsilon \Box \neg isfull(c) \Box \Box(c \Box c^{\Box} \Box \varepsilon)$ (A21, A22, R3)

where $c^{\Box} \Box c \cdot \Box x\Box$ is a constant. If the predicate '$isfull(c)$' is true, the normal form of $\mathtt{put}(c, x)$ is $\Box\varepsilon$; otherwise, its normal form is $\Box(c \Box c^{\Box} \Box \varepsilon)$.

(4) $\mathtt{get}(c, x)$:

$\mathtt{get}(c, x)$
$= \mathtt{get}(c, x) \Box \mathsf{true}$ (A22, R3)
$= \mathtt{get}(c, x) \Box (isempty(c) \Box \neg isempty(c))$ (A22, R3)
$= \mathtt{get}(c, x) \Box isempty(c) \Box \mathtt{get}(c, x) \Box \neg isempty(c)$ (A22, R3)
$= isempty(c) \Box \mathsf{skip} \Box \neg isempty(c) \Box x :\Box head(c) \Box c :\Box tail(c)$ (A29, A30)
$= isempty(c) \Box \Box \varepsilon \Box \neg isempty(c) \Box \Box(x \Box head(c) \Box c \Box c^{\Box} \Box \varepsilon)$ (A21, A22, R3)

where $c^{\Box} \Box tail(c)$ is a constant. If the predicate '$isempty(c)$' is true, the normal form of $\mathtt{get}(c, x)$ is $\Box\varepsilon$; or else, its normal form is $\Box(x \Box head(c) \Box c \Box c^{\Box} \Box \varepsilon)$. □

Further, by Theorem 2.1, we know that except asynchronous communication commands, other statements in MSVL can also be deduced into their normal forms by axioms in Table 4. Then combing Theorem 3.1 and Theorem 2.1, we can obtain the following corollary.

Corollary 3.1 *Any program P in MSVL with asynchronous communication commands can be deduced into normal form by state axioms and state inference rules in Tables 4 and 6.*

The following example illustrates how to use asynchronous communication axioms. Thereinto, we assume the maximal capacity of the channel c is 3.

Example 1

$$
\begin{array}{ll}
 & c \Box \Box\Box \Box (\texttt{send}(c, 5) \parallel \texttt{receive}(c, x)) \\
\exists & c \Box \Box\Box \Box (c :\Box \Box 5\Box \parallel \texttt{receive}(c, x)) \qquad \text{(A23, R1)} \\
\exists & c \Box \Box\Box \Box (c :\Box \Box 5\Box \parallel \Box (\mathsf{lbf}(c) \Box \texttt{receive}(c, x))) \qquad \text{(A26, R1)} \\
\exists & c \Box \Box\Box \Box (\Box(c \Box \Box 5\Box \Box \varepsilon) \parallel \Box (\mathsf{lbf}(c) \Box \texttt{receive}(c, x))) \qquad \text{(A21, R1)} \\
\exists & c \Box \Box\Box \Box \Box(c \Box \Box 5\Box \Box \varepsilon \parallel \mathsf{lbf}(c) \Box \texttt{receive}(c, x)) \qquad \text{(A7, A10, A22, R1, R3)}
\end{array}
$$

Thus, the original program has been in its normal form.

Equipped with the state axioms (A23–30), the extended axiomatic system of MSVL can be used to verify distributed systems communicating by channels. Then we give a simple example to show how it works. We use P_f^i to denote the program at the state s_i with the present component P_c^i and the future component $\Box P_f^{i+1}$.

Example 2 The MSVL program is

$$P \stackrel{\text{def}}{\Box} c \Box \Box\Box \Box (\texttt{send}(c, 5) \parallel \texttt{receive}(c, x))$$

and the desired property is 'x is equal to 5 finally'. Let $p \stackrel{\text{def}}{\Box} x \Box 5$. Then the property is formalized as the PPTL formula

$$A \stackrel{\text{def}}{\Box} \Box(\varepsilon \rightarrow p)$$

The triple to deduce is

$$\{A\}\ P\ \{B\} \Box \{\sigma_0, \Box(\varepsilon \rightarrow p)\}\ P\ \{\sigma_2, p\} \tag{1}$$

The verification proceeds as follows: by Example 1, we know that $P \exists P_c^0 \Box \Box P_f^1$, where $P_c^0 \exists c \Box \Box\Box$ and $P_f^1 \exists c \Box \Box 5\Box \Box \varepsilon \parallel \mathsf{lbf}(c) \Box \texttt{receive}(c, x)$. Further, $\Box(\varepsilon \rightarrow p) \equiv p \Box \varepsilon \Box \Box\Box(\varepsilon \rightarrow p)$. Then

$$(1)\ \Box\Box\ \{\sigma_0, p \Box \varepsilon \Box \Box\Box(\varepsilon \rightarrow p)\}\ P_c^0 \Box \Box P_f^1\ \{\sigma_2, p\} \qquad \text{(A7,A10,A21–23,A26,R1,R3,ISR)} \tag{2}$$

It is obvious that $c \Box \Box\Box \rightarrow \text{true}$. Hence $P_c^0 \rightarrow \text{true}$, that is the present component of the property is satisfied at state s_0. Then by rule (ANext), we can obtain:

$$(2)\ \Box\Box\ \{\sigma_0, \text{true}\}\ P_c^0\ \{\sigma_0, \text{true}\}\ \text{and}\ \{\sigma_1, \Box(\varepsilon \rightarrow p)\}\ P_f^1\ \{\sigma_2, p\} \qquad \text{(ANext)}$$

By axiom (APC), it is easy to see that $\{\sigma_0, \text{true}\}\ P_c^0\ \{\sigma_0, \text{true}\}$ holds. Thus, we just need to prove $\{\sigma_1, \Box(\varepsilon \rightarrow p)\}\ P_f^1\ \{\sigma_2, p\}$ holds. At state s_1, since the program P_f^1 can be deduced as:

$$
\begin{array}{ll}
 & c \Box \Box 5\Box \Box \varepsilon \parallel \mathsf{lbf}(c) \Box \texttt{receive}(c, x) \\
\exists & c \Box \Box 5\Box \Box \mathsf{lbf}(c) \Box \texttt{receive}(c, x) \qquad \text{(A8–10,A22,A25,R1)} \\
\exists & c \Box \Box 5\Box \Box p_c \Box \texttt{receive}(c, x) \qquad \text{(A1,R1)} \\
\exists & c \Box \Box 5\Box \Box p_c \Box x :\Box \mathit{head}(c) \Box c :\Box \mathit{tail}(c) \qquad \text{(A25,R1)} \\
\exists & c \Box \Box 5\Box \Box p_c \Box \Box(x \Box 5 \Box c \Box \Box\Box \Box \varepsilon) \qquad \text{(A21,A22,R1)}
\end{array}
$$

Then according to rule (ISR), we have:

$$\{\sigma_1, \Box(\varepsilon \rightarrow p)\}\ P_f^1\ \{\sigma_2, p\}\ \Box\Box\ \{\sigma_1, p \Box \varepsilon \Box \Box\Box(\varepsilon \rightarrow p)\}\ P_c^1 \Box \Box P_f^2\ \{\sigma_2, p\} \qquad \text{(ISR)} \tag{3}$$

where $P_c^1 \exists c \Box \Box 5\Box \Box p_c$ and $P_f^2 \exists x \Box 5 \Box c \Box \Box\Box \Box \varepsilon$. As $c \Box \Box 5\Box \Box p_c \rightarrow \text{true}$, the present component of program satisfies the present component of property at state s_1. By axiom (ANext),

$$(3)\ \Box\Box\ \{\sigma_1, \text{true}\}\ P_c^1\ \{\sigma_1, \text{true}\}\ \text{and}\ \{\sigma_2, \Box(\varepsilon \rightarrow p)\}\ P_f^2\ \{\sigma_2, p\} \qquad \text{(ANext)} \tag{4}$$

According to axiom (APC), we know that $\{\sigma_1, \text{true}\}\ P_c^1\ \{\sigma_1, \text{true}\}$ holds. Then we merely need to prove

$$\{\sigma_2, \Box(\varepsilon \rightarrow p)\}\ P_f^2\ \{\sigma_2, p\}\ \Box\Box\ \{\sigma_2, p \Box \varepsilon \Box \Box\Box(\varepsilon \rightarrow p)\}\ P_e \Box \varepsilon\ \{\sigma_2, p\} \qquad \text{(ISR)} \tag{5}$$

where $P_e \exists x \Box 5 \Box c \Box \Box\Box$. Since $x \Box 5 \Box c \Box \Box\Box \rightarrow x \Box 5$, namely $P_e \rightarrow p$, the present component of program satisfies the present component of property at state s_2. Therefore, by axiom (AEmpty), we have

$$(5)\ \Box\Box\ \{\sigma_2, \varepsilon\}\ \varepsilon\ \{\sigma_2, \varepsilon\}\ \text{and}\ \{\sigma_2, p\}\ P_e\ \{\sigma_2, p\} \qquad \text{(AEmpty)}$$

According to rule (AEM) and (APC), $\{\sigma_2, \varepsilon\}\ \varepsilon\ \{\sigma_2, \varepsilon\}$ and $\{\sigma_2, p\}\ P_e\ \{\sigma_2, p\}$ are satisfied. Thus, the property $\Box(\varepsilon \rightarrow p)$ can be satisfied by the program P.

3.3. Soundness and completeness

In this subsection, we prove the soundness and completeness of asynchronous communication axioms. Theorem 3.2 tells us that asynchronous communication axioms are sound, that is axioms (A23–30) also hold semantically in the model theory.

Theorem 3.2 (Soundness of asynchronous communication axioms) *Axioms (A23–30) are sound, that is* $P = Q \Leftrightarrow P \equiv Q$.

Proof. Now we demonstrate that axioms (A23–30) in Table 6 are valid in the model theory. Note that the logic law (L6) is used implicitly.

Axiom (A23): We need to prove $\mathtt{send}(c, x) \equiv c :\Box c \cdot \lfloor x \rfloor$ when $isfull(c)$ is false.

$$
\begin{array}{ll}
 & \mathtt{send}(c, x) \\
\equiv & \mathsf{await}(\neg isfull(c))\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad \text{(definition of send)} \\
\equiv & (\neg isfull(c) \Box \varepsilon)\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad \text{(L5, } isfull(c) \text{ is false)} \\
\equiv & \varepsilon\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad (\neg isfull(c) \text{ is true)} \\
\equiv & c :\Box c \cdot \lfloor x \rfloor \quad \text{(L1)}
\end{array}
$$

Axiom (A24): We need to prove $\mathtt{send}(c, x) \equiv \Box(\mathsf{lbf}(c) \Box \mathtt{send}(c, x))$ when $isfull(c)$ is true.

$$
\begin{array}{ll}
 & \mathtt{send}(c, x) \\
\equiv & \mathsf{await}(\neg isfull(c))\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad \text{(definition of send)} \\
\equiv & isfull(c) \Box \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isfull(c)))\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad \text{(L5, } isfull(c) \text{ is true))} \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isfull(c)))\ ;\ c :\Box c \cdot \lfloor x \rfloor \quad (isfull(c) \text{ is true)}) \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isfull(c))\ ;\ c :\Box c \cdot \lfloor x \rfloor) \quad \text{(L3)} \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathtt{send}(c, x)) \quad \text{(L2, definition of send)}
\end{array}
$$

Axiom (A25): We need to prove $\mathtt{receive}(c, x) \equiv x :\Box head(c) \Box c :\Box tail(c)$ when $isempty(c)$ is false.

$$
\begin{array}{ll}
 & \mathtt{receive}(c, x) \\
\equiv & \mathsf{await}(\neg isempty(c))\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad \text{(definition of receive)} \\
\equiv & (\neg isempty(c) \Box \varepsilon)\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad \text{(L5, } isempty(c) \text{ is false)} \\
\equiv & \varepsilon\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad (\neg isempty(c) \text{ is true)} \\
\equiv & x :\Box head(c) \Box c :\Box tail(c) \quad \text{(L1)}
\end{array}
$$

Axiom (A26): We need to prove $\mathtt{receive}(c, x) \equiv \Box(\mathsf{lbf}(c) \Box \mathtt{receive}(c, x))$ when $isempty(c)$ is true.

$$
\begin{array}{ll}
 & \mathtt{receive}(c, x) \\
\equiv & \mathsf{await}(\neg isempty(c))\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad \text{(definition of receive)} \\
\equiv & isempty(c) \Box \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isempty(c)))\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad \text{(L5, } isempty(c) \text{ is true)} \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isempty(c)))\ ;\ x :\Box head(c) \Box c :\Box tail(c) \quad (isempty(c) \text{ is true)} \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathsf{await}(\neg isempty(c))\ ;\ x :\Box head(c) \Box c :\Box tail(c)) \quad \text{(L3)} \\
\equiv & \Box(\mathsf{lbf}(c) \Box \mathtt{receive}(c, x)) \quad \text{(L2, definition of receive)}
\end{array}
$$

Axiom (A27): We need to prove $\mathtt{put}(c, x) \equiv c :\Box c \cdot \lfloor x \rfloor$ when $isfull(c)$ is false.

$$
\begin{array}{ll}
 & \mathtt{put}(c, x) \\
\equiv & \mathtt{if}\ (\neg isfull(c))\ \mathtt{then}\ \{c :\Box c \cdot \lfloor x \rfloor\}\ \mathtt{else}\ \{\mathsf{skip}\} \quad \text{(definition of put)} \\
\equiv & \neg isfull(c) \Box c :\Box c \cdot \lfloor x \rfloor \Box isfull(c) \Box \mathsf{skip} \quad \text{(definition of if)} \\
\equiv & c :\Box c \cdot \lfloor x \rfloor \quad (isfull(c) \text{ is false)}
\end{array}
$$

Axiom (A28): We need to prove $\mathtt{put}(c, x) \equiv \mathsf{skip}$ when $isfull(c)$ is true.

$$
\begin{array}{ll}
 & \mathtt{put}(c, x) \\
\equiv & \mathtt{if}\ (\neg isfull(c))\ \mathtt{then}\ \{c :\Box c \cdot \lfloor x \rfloor\}\ \mathtt{else}\ \{\mathsf{skip}\}) \quad \text{(definition of put)} \\
\equiv & \neg isfull(c) \Box c :\Box c \cdot \lfloor x \rfloor \Box isfull(c) \Box \mathsf{skip} \quad \text{(definition of put)} \\
\equiv & \mathsf{skip} \quad (isfull(c) \text{ is true)}
\end{array}
$$

Axiom (A29): We need to prove $\texttt{get}(c, x) \equiv x :\Box\ head(c) \Box c :\Box\ tail(c)$ when $isempty(c)$ is false.

$$
\begin{array}{ll}
& \texttt{get}(c, x) \\
\equiv & \texttt{if } (\neg isempty(c)) \texttt{ then } \{x :\Box\ head(c) \Box c :\Box\ tail(c)\} \texttt{ else } \{\texttt{skip}\} \quad \text{(definition of get)} \\
\equiv & \neg isempty(c) \Box x :\Box\ head(c) \Box c :\Box\ tail(c) \Box isempty(c) \Box \texttt{skip} \quad \text{(definition of if)} \\
\equiv & x :\Box\ head(c) \Box c :\Box\ tail(c) \quad (isempty(c) \text{ is false})
\end{array}
$$

Axiom (A30): We need to prove $\texttt{get}(c, x) \equiv \texttt{skip}$ when $isempty(c)$ is true.

$$
\begin{array}{ll}
& \texttt{get}(c, x) \\
\equiv & \texttt{if } (\neg isempty(c)) \texttt{ then } \{x :\Box\ head(c) \Box c :\Box\ tail(c)\} \texttt{ else } \{\texttt{skip}\} \quad \text{(definition of get)} \\
\equiv & \neg isempty(c) \Box x :\Box\ head(c) \Box c :\Box\ tail(c) \Box isempty(c) \Box \texttt{skip} \quad \text{(definition of if)} \\
\equiv & \texttt{skip} \quad (isempty(c) \text{ is true})
\end{array}
$$

Therefore, axioms (A23–30) are valid in the model theory. □

Lemma 3.1 claims that asynchronous communication commands can be semantically rewritten into their normal forms, which is the counterpart of Theorem 3.1 in the model theory.

Lemma 3.1 *Statements* $\texttt{send}(c, x)$, $\texttt{receive}(c, x)$, $\texttt{put}(c, x)$ *and* $\texttt{get}(c, x)$ *can be reduced into normal form in the model theory.*

The proof can be found in the Appendix. From the proofs of Lemma 3.1 and Theorem 3.1, we find that asynchronous communication commands can be transformed into the same normal forms by means of both reduction and deduction.

Theorem 3.3 asserts the extended state axioms and inference rules are complete, that is if any program P in MSVL with asynchronous communication commands can be reduced into Q in the model theory, it can also be deduced into Q in axiomatic semantics by state axioms and inference rules in Tables 4 and 6.

Theorem 3.3 *Let P and Q be programs in MSVL with asynchronous communication commands, state axioms and inference rules in Tables 4 and 6 are complete, that is $P \equiv Q$ $\Box\Box$ $P \boxminus Q$.*

Proof. In Theorem 2.1, all the axioms excluding the asynchronous communication axioms have been proved to be complete for MSVL programs without asynchronous communication commands. Thus, we merely need to prove the cases for asynchronous communication commands. Firstly, in accordance with the proof of Lemma 3.1, we can clearly obtain:

(1) $\texttt{send}(c, x) \equiv c :\Box\ c \cdot \lceil x \rceil$
$\Box\Box\ \texttt{send}(c, x) \boxminus c :\Box\ c \cdot \lceil x \rceil$ if $isfull(c)$ is false

(2) $\texttt{send}(c, x) \equiv \Box(\texttt{lbf}(c) \Box \texttt{send}(c, x))$
$\Box\Box\ \texttt{send}(c, x) \boxminus \Box(\texttt{lbf}(c) \Box \texttt{send}(c, x))$ if $isfull(c)$ is true

(3) $\texttt{receive}(c, x) \equiv x :\Box\ head(c) \Box c :\Box\ tail(c)$
$\Box\Box\ \texttt{receive}(c, x) \boxminus x :\Box\ head(c) \Box c :\Box\ tail(c)$ if $isempty(c)$ is false

(4) $\texttt{receive}(c, x) \equiv \Box(\texttt{lbf}(c) \Box \texttt{receive}(c, x))$
$\Box\Box\ \texttt{receive}(c, x) \boxminus \Box(\texttt{lbf}(c) \Box \texttt{receive}(c, x))$ if $isempty(c)$ is true

(5) $\texttt{put}(c, x) \equiv c :\Box\ c \cdot \lceil x \rceil$
$\Box\Box\ \texttt{put}(c, x) \boxminus c :\Box\ c \cdot \lceil x \rceil$ if $isfull(c)$ is false

(6) $\texttt{put}(c, x) \equiv \texttt{skip}$
$\Box\Box\ \texttt{put}(c, x) \boxminus \texttt{skip}$ if $isfull(c)$ is true

(7) $\texttt{get}(c, x) \equiv x :\Box\ head(c) \Box c :\Box\ tail(c)$
$\Box\Box\ \texttt{get}(c, x) \boxminus x :\Box\ head(c) \Box c :\Box\ tail(c)$ if $isempty(c)$ is false

(8) $\texttt{get}(c, x) \equiv \texttt{skip}$
$\Box\Box\ \texttt{get}(c, x) \boxminus \texttt{skip}$ if $isempty(c)$ is true

Further, on one hand, by Lemma 3.1, we know that any asynchronous communication command P can be rewritten into its normal form Q in the model theory. On the other hand, according to Theorem 3.1, P can also be deduced into its normal form R in axiomatic semantics. From the proofs, we can see that R is equivalent to Q, i.e. $R \equiv Q$. During the transformation of P into Q in the model theory, each logic law has a corresponding axiom in the axiomatic system. That is, state axioms (A8,A9,A7,R3,A20) and (R1) correspond to logic laws (L1–L6) respectively. Therefore, $P \equiv Q$ $\Box\Box$ $P \boxminus Q$ holds for any asynchronous communication command P.

Table 7. Some useful theorems

(T1)	$\varepsilon \parallel P \Leftrightarrow P$ (T2) $\Box (P_1 \parallel P_2) \Leftrightarrow \Box P_1 \parallel \Box P_2$	
(T3)	$(w_1 \Box \Box P_1) \parallel (w_2 \Box \Box P_2) \Leftrightarrow w_1 \Box w_2 \Box \Box (P_1 \parallel P_2)$	
(T4)	$(w_1 \Box P_1) \parallel (w_2 \Box P_2) \Leftrightarrow w_1 \Box w_2 \Box (P_1 \parallel P_2)$	
(T5)	$\mathsf{lbf}(V_b) \Box \mathsf{frame}(V_b) \Box \mathsf{await}(b) \Leftrightarrow \mathsf{lbf}(V_b) \Box \Box (\mathsf{lbf}(V_b) \Box \mathsf{frame}(V_b) \Box \mathsf{await}(b))$	if b is false
(T6)	$\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{send}(c, x) \Leftrightarrow \mathsf{lbf}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{send}(c, x))$	if *isfull*(c) is true
(T7)	$\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{receive}(c, x) \Leftrightarrow \mathsf{lbf}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{receive}(c, x))$	if *isempty*(c) is true

Further, combing Theorem 2.1 we can obtain that $P \equiv Q \Box\Box P \Leftrightarrow Q$ for any MSVL program with asynchronous communication commands. □

Since asynchronous communication axioms are concerned with state deduction, the introduction of asynchronous communication axioms into the original axiomatic semantics of MSVL [YDM10] do not alter the soundness and relative completeness of the interval rules. Thus, the extended axiomatic system is sound and relatively complete.

3.4. Some theorems

Based on state axioms within the extended axiomatic system, we can obtain some useful theorems listed in Table 7, where w_1 and w_2 are state formulas. Theorems (T1–T4) concerns the parallel statement. Particularly, (T1) tells us that any program P in parallel with the termination statement can be deduced into itself. (T2) states the distributive property of $\Box$ over ||, which can be considered as an instance of state axiom (A22). Further, (T3) says that if both the two parallel programs have been in their normal forms, the parallel relation can be transferred to the future components. As an extension of (T3), (T4) declares that if state formulas appear in a parallel statement, they can be conjuncted with the remaining program that is in parallel. Theorem (T5) deals with the conjunction of 'lbf', 'frame' and 'await' when the boolean expression b cannot be satisfied. Moreover, (T6) is for the conjunction of 'lbf', 'frame' and '`send`' when the channel c is full while (T7) is for the conjunction of 'lbf', 'frame' and '`receive`' when the channel c is empty.

Here we only demonstrate the formal proofs of (T3) and (T6). Others can be proved in a similar way.

(T3):

$(w_1 \Box \Box P_1) \parallel (w_2 \Box \Box P_2)$

$\Leftrightarrow (w_1 \Box \Box P_1 ; \mathsf{true}) \Box w_2 \Box \Box P_2 \Box w_1 \Box \Box P_1 \Box (w_2 \Box \Box P_2 ; \mathsf{true})$ (A10)

$\Leftrightarrow w_1 \Box \Box (P_1 ; \mathsf{true}) \Box w_2 \Box \Box P_2 \Box w_1 \Box \Box P_1 \Box w_2 \Box \Box (P_2 ; \mathsf{true})$ (A7,A9)

$\Leftrightarrow w_1 \Box w_2 \Box \Box ((P_1 ; \mathsf{true}) \Box P_2) \Box w_1 \Box w_2 \Box \Box (P_1 \Box (P_2 ; \mathsf{true}))$ (A22,R3)

$\Leftrightarrow w_1 \Box w_2 \Box \Box (P_1 \parallel P_2)$ (A10,A22,R3)

(T6):

$\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{send}(c, x)$

$\Leftrightarrow \mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathtt{send}(c, x))$ (*isfull*(c) is true, A24)

$\Leftrightarrow \mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathtt{send}(c, x)) \Box \mathsf{more}$ (A11)

$\Leftrightarrow \mathsf{lbf}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathsf{frame}(c)) \Box \Box (\mathsf{lbf}(c) \Box \mathtt{send}(c, x))$ (A6)

$\Leftrightarrow \mathsf{lbf}(c) \Box \Box (\mathsf{lbf}(c) \Box \mathsf{frame}(c) \Box \mathtt{send}(c, x))$ (A22,R3)

4. An application: Ricart–Agrawala algorithm

In this section, we apply the extended axiomatic system of MSVL to verify distributed systems in practice. We will employ the RA [RA81] algorithm, which is a well-known distributed mutual exclusion algorithm. Firstly, RA algorithm is modeled by MSVL; secondly, its desired properties are specified by PPTL; then whether or not the algorithm satisfies a property is verified with the extended axiomatic system of MSVL.

4.1. Description

Ricart–Agrawala algorithm is designed by Glenn Ricart and Ashok Agrawala in 1981 to guarantee processes mutually access the critical section (CS) in a distributed system. It is a permission-based method in the sense that a process can enter the CS if and only if it has obtained the permissions from all the other processes in a distributed system. In RA algorithm, all processes have distinct numeric ID and maintain logical clocks by themselves. Further, first-in-first-out communication channels are used to connect any two processes. The number of message passing for per invocation of CS is 2*(N-1) and less than that of Lamport' Distributed Bakery algorithm. In RA algorithm, all the processes in the network behave in the same manner: requesting, executing and releasing CS. A brief introduction to RA algorithm is as follows and the details can refer to [RA81]. For each process i,

Requesting the critical section:

1. When a process i wants to enter the CS, it sends a REQUEST message with time-stamp to all the processes.
2. When process j receives the REQUEST message from process i, it sends a REPLY message to process i if j is neither requesting nor executing the CS or if j is requesting and i's request's time-stamp is smaller than j's own request's time-stamp. Otherwise the request is deferred.

Executing the critical section:

1. Process i enters the CS after it has received REPLY messages from all the processes.

Releasing the critical section:

1. When process i exits the CS, it sends REPLY messages to all the deferred requests.

We need to illustrate more about the condition 'i's request's time-stamp is smaller than j's own request's time-stamp'. This condition covers two cases: 1. the time-stamp of the request of process i is strictly smaller than that of process j; 2. the time-stamp of the request of process i is equal to that of process j, but the numeric ID i is less than the numeric ID j. Formally, let ts_i and ts_j denote the time-stamps of processes i and j. Then it exactly means $ts_i < ts_j \Box ts_i \Box ts_j \Box i < j$. In this case, we call process i has a higher priority than process j. With the execution of RA algorithm, the time-stamp will become increasingly larger and larger. As a result, processes operate on unbounded local variables ranging over the natural numbers. Thus, RA algorithm has an infinite state space.

4.2. Modeling

At first, we model RA algorithm with MSVL. Then the general program P_{RA} of RA algorithm for arbitrary N processes is displayed in Fig. 2. For a clear presentation, we write lbf(...) and frame(...) short for lbf($c_{12}, \ldots, c_{1N}, \ldots, c_{N1}, \ldots, c_{N(N-1)}, ts, hts, rd, R, req, result, max, At\_cr$) and frame($c_{12}, \ldots, c_{1N}, \ldots, c_{N1}, \ldots, c_{N(N-1)}, ts, hts, rd, R, req, result, max, At\_cr$) respectively. Hence, the program can be written as follows:

$$P_{RA} \stackrel{\text{def}}{\Box} \text{frame}(\ldots) \Box (w \Box \text{skip} \,;\, \Box_{i\Box 1}^{N} (P_i^1 \,||\, P_i^2))$$

where w is short for the underlined part; P_i^1 stands for the sub-program '`while`(true) `do`{$req[i] :\Box 1 \Box R[i, i] :\Box 1 ; \ldots ; \Box\ l : \{l :\Box 1$;`while`($l \leq N$) `do`{`if`($rd[i, l] \Box 1$) `then`{$rd[i, l] :\Box 0$;`send`($c_{il}, -1$)} `else` {skip} ;$l :\Box l + 1$}}}'; P_i^2 represents the sub-program '`while`(true) `do`{$\Box_{t\Box 1, \Box i}^{N}$(`receive`($c_{ti}$, $result[i, t]$) ; ... {$rd[i, t] :\Box 1$})}'. This program is mainly composed of the initialization of variables and the parallel execution of N processes. Further, for each process i, its execution can be regarded as a parallel combination of a sending sub-process P_i^1 to request the CS and a receiving sub-process P_i^2 to receive requests and determine whether to reply or defer the reply. In RA algorithm, since communication is peer-to-peer and a channel indicates the exact identities of processes, we directly denote REQUEST message by the time-stamp, namely an integer equal to or greater than 0. For the other kind of message, i.e. REPLY messages, we signify them by -1. In this way, the two sorts of messages can be told apart in terms of their numeric values.

$$
\begin{array}{l}
\mathsf{frame}(c_{12},\ldots,c_{1N},\ldots,c_{N1},\ldots,c_{N(N-1)},ts,hts,rd,R,req,result,max,At\_cr) \\
(\mathsf{chn}\ c_{12}(N),\ldots,c_{1N}(N),\ldots,c_{N1}(N),\ldots,c_{N(N-1)}(N)\quad \mathsf{int}\ ts[N+1]=\{0,\ldots,0\}, \\
\underline{hts[N+1,N+1]=\{0,\ldots,0\},max[N+1]=\{0,\ldots,0\},result[N+1,N+1]} \\
\underline{\mathsf{bool}\ rd[N+1,N+1]=\{0,\ldots,0\},R[N+1,N+1]=\{0,\ldots,0\},} \\
\underline{req[N+1]=\{0,\ldots,0\},At\_cr[N+1]=\{0,\ldots,0\}}\quad \mathsf{skip}\ ; \qquad /\ \text{initialization}\ / \\
\bigwedge_{i=1}^{N}\Bigg(P_i^1 \Bigg\{ \begin{array}{l}
\mathtt{while}\ (\mathsf{true})\ \mathtt{do}\ \{ \qquad /\ \text{sending sub-process to request CS}\ / \\
\quad req[i]:=1 \quad R[i,i]:=1\ ; \\
\quad\ l:\{l:=1\ ; \\
\qquad \mathtt{while}\ (l\le N)\ \mathtt{do}\ \{ \\
\qquad\quad \mathtt{if}\ (max[i]<hts[i,l])\ \mathtt{then}\ \{max[i]:=hts[i,l]\}\ \mathtt{else}\ \{\mathsf{skip}\}\ ; \\
\qquad\quad l:=l+1\}\}\ ; \\
\quad ts[i]:=max[i]+1\ ; \\
\quad \bigwedge_{j=1,\neq i}^{N}\mathsf{send}(c_{ij},ts[i])\ ; \qquad /\ \text{i does not send to itself}\ / \\
\quad \mathsf{await}(R[i,1]=1 \quad\ldots\quad R[i,N]=1)\ ; \\
\quad At\_cr[i]:=1\ ; \\
\quad At\_cr[i]:=0 \quad req[i]:=0 \quad R[i,1]:=0 \quad\ldots\quad R[i,N]:=0\ ; \\
\quad\ l:\{l:=1\ ; \\
\qquad \mathtt{while}\ (l\le N)\ \mathtt{do}\ \{ \\
\qquad\quad \mathtt{if}\ (rd[i,l]=1)\ \mathtt{then}\ \{rd[i,l]:=0;\mathsf{send}(c_{il},-1)\}\ \mathtt{else}\ \{\mathsf{skip}\}; \\
\qquad\quad l:=l+1\}\}\quad \}
\end{array} \\
\qquad\qquad P_i^2 \Bigg\{ \begin{array}{l}
\mathtt{while}\ (\mathsf{true})\ \mathtt{do}\ \{ \qquad /\ \text{receiving sub-process}\ / \\
\quad \bigwedge_{t=1,\neq i}^{N}\Bigg(\begin{array}{l}
\mathtt{receive}(c_{ti},result[i,t])\ ; \\
\mathtt{if}\ (result[i,t]=-1)\ \mathtt{then}\ \{R[i,t]:=1\} \\
\mathtt{else}\ \{ \\
\quad \mathtt{if}\ (result[i,t]>hts[i,t])\ \mathtt{then}\ \{hts[i,t]:=result[i,t]\}\ \mathtt{else}\ \{\mathsf{skip}\} \\
\quad \mathtt{if}\ (req[i]=0 \quad result[i,t]<ts[i] \quad (result[i,t]=ts[i] \quad t<i)) \\
\quad \mathtt{then}\ \{\mathsf{send}(c_{it},-1)\} \\
\quad \mathtt{else}\ \{rd[i,t]:=1\}\}
\end{array}\Bigg) \\
\}
\end{array} \Bigg)
\end{array}
$$

Fig. 2. The MSVL program for Ricart–Agrawala algorithm with arbitrarily many processes

In P_{RA}, for any array variable x, process i only has $x[i]$ and cannot access $x[j]$ ($j \neq i$) owned by other processes. That is, $x[i]$ can be regarded as a local variable possessed by i. The meaning of the involved variables is explained in the following, where $1 \le i, j \le N$, $j \neq i$. Thereinto, the values of ts, hts and max belong to the set of non-negative integers and the type of req, rd, R and At_cr is boolean.

- ts: time-stamp number. $ts[i]$ indicates the time-stamp of the request of process i.
- hts: the highest time-stamp number of a process. $hts[i, j]$ represents the highest time-stamp number of process j known by i.
- req: requesting the CS flag. $req[i] = 1$ signifies process i is requesting the CS.
- c_{ij}: a unidirectional channel from process i to process j with the capacity of N, through which i sends messages to j and j receives messages from i.
- rd: request-deferred flag. $rd[i, j] = 1$ denotes process i defers the REPLY message to the request of process j.
- max: the maximal time-stamp number among all the received requests. $max[i]$ refers to the maximum time-stamp number among the requests of all the other processes that received by i.
- $result$: storing array; each element is an integer greater than or equal to -1. $result[i, j]$ is used to store the received message by process i from process j via the channel c_{ji}.
- R: receiving REPLY message flag. $R[i, j] = 1$ implies that process i has received the REPLY message from process j.
- At_cr: using the CS flag. $At\_cr[i] = 1$ states that process i is using the CS at present.

4.3. Properties

RA algorithm is an effective approach to implement the mutual exclusion for distributed systems. Generally, RA algorithm has the following characteristics:

1. Mutual exclusion: At any instant, only one process can access the CS.
2. Freedom of starvation: As long as a process requests to enter the CS, it should not wait endlessly and will obtain the CS eventually.
3. First-come-first-served (FCFS): The execution of the requests of processes is always in the order of their time-stamps. Particularly when the time-stamps are equal to each other, the process with the lower numeric identity enters the CS earlier.
4. A process enters the CS only if it gets permissions from all the other processes in the system.
5. If a process enters the CS, it has a higher priority than all the other processes.
6. A process does not posses the CS when it has not requested.
7. After a process has released the CS, no other processes can enter the CS in less than a one-way trip transmission time.
8. Once the request of process i has been dealt with by all the other processes, no other process enters the CS twice before i enters it.

Mutual exclusion and freedom of starvation are basic properties while other properties are special properties of RA algorithm. In particular, the most distinguished feature of RA algorithm is the FCFS property, which achieves fairness. Actually, the properties of FCFS and starvation-free can guarantee the property of deadlock-free. To specify these properties with PPTL, the following propositions are introduced:

$$r_i \stackrel{\text{def}}{=} At\_cr[i] \sqsupset 1 \qquad q_i \stackrel{\text{def}}{=} req[i] \sqsupset 1 \qquad s_{ij} \stackrel{\text{def}}{=} R[i,j] \sqsupset 1$$
$$p_{ij} \stackrel{\text{def}}{=} ts[i] \leq ts[j] \qquad t_{ij} \stackrel{\text{def}}{=} rd[i,j] \sqsupset 1 \qquad m_{ij} \stackrel{\text{def}}{=} head(c_{ij}) \sqsupset -1$$

where r_i represents the process i is using the CS; q_i denotes process i is requesting the CS; s_{ij} signifies process i has received the REPLY message from process j; p_{ij} stands for the time-stamp of process i is equal to or less than that of process j while $\neg p_{ij}$ stands for the time-stamp of process i is greater than that of process j; t_{ij} states process i defers the REPLY message to the request of process j; m_{ij} indicates the first message in the channel c_{ij} is the REPLY message. Then we can describe the above properties as follows:

1. Mutual exclusion: any two processes cannot be in the CS at the same time.

$$\bigwedge_{i\sqsupset 1}^{N} \bigwedge_{j\sqsupset 1, \sqsupset i}^{N} \Box\neg(r_i \sqsupset r_j)$$

2. Freedom of starvation:

$$\bigwedge_{i\sqsupset 1}^{N} \Box(q_i \rightarrow \Diamond r_i)$$

3. First-come-first-served property:

$$\bigwedge_{i\sqsupset 1}^{N} \bigwedge_{j>i}^{N} \Box(p_{ij} \sqsupset \Diamond(r_i \sqsupset \mathsf{skip} ; \Diamond r_j) \sqsupset \neg p_{ij} \sqsupset \Diamond(r_j \sqsupset \mathsf{skip} ; \Diamond r_i))$$

For two processes i and j with $i < j$, if the requesting time-stamp of process i is smaller than or equal to that of process j, i will be served and enter the CS earlier than j; otherwise, j will obtain the CS earlier.

4. If a process enters the CS, it must have acquired permissions from all the other processes in the system.

$$\bigwedge_{i\sqsupset 1}^{N} \Box\left(r_i \rightarrow \bigwedge_{j\sqsupset 1}^{N} s_{ij}\right)$$

5. If a process enters the CS, its time-stamp is less than or equal to the time-stamps of those processes that have higher numeric identities and less than the time-stamps of those processes that have lower numeric identities.

$$\bigwedge_{i\Box 1}^{N} \Box \left(r_i \rightarrow \left(\bigwedge_{j>i}^{N} p_{ij} \Box \bigwedge_{j\Box 1}^{i-1} \neg p_{ji} \right) \right)$$

6. If a process does not request the CS, it will not get the REPLY message from other processes.

$$\bigwedge_{i\Box 1}^{N} \Box \left(\neg q_i \rightarrow \bigwedge_{j\Box 1, \Box i}^{N} \neg s_{ij} \right)$$

7. At any instant, none of processes can enter the CS at the next state after a process releases the CS.

$$\bigwedge_{i\Box 1}^{N} \Box \left(r_i \Box \Box \neg r_i \rightarrow \neg \Box^2 \left(\bigvee_{j\Box 1}^{N} r_j \right) \right)$$

8. For the request of process i, if all the other processes j either send a REPLY message to i or defer the request, no other process k can enter the CS twice before i enters it.

$$\bigwedge_{i\Box 1}^{N} \Box \left(\bigwedge_{j\Box 1, \Box i}^{N} (m_{ji} \Box t_{ji}) \rightarrow \neg \bigvee_{k\Box 1, \Box i}^{N} (\Diamond(r_k \Box \mathsf{skip}) ; \Diamond(r_k \Box \mathsf{skip}) ; \Diamond(r_i)) \right)$$

4.4. Verification

This subsection presents a minutely verifying procedure for an instance of two processes. Note that even if for two processes, we still need to verify an infinite state system due to the unboundedness of time-stamps. Let $N \Box 2$. With the modeling method, the MSVL program for two processes can be obtained as

$$P_{RA2} \stackrel{\text{def}}{\Box} \mathsf{frame}(\ldots) \Box (w \Box \mathsf{skip} ; (P_1^1 \parallel P_1^2) \parallel (P_2^1 \parallel P_2^2))$$

in Fig. 3, where $At\_cr[1]$, $req[1]$, $R[1, 1]$, $R[1, 2] :\Box 0$ means $At\_cr[1] :\Box 0 \Box req[1] :\Box 0 \Box R[1, 1] :\Box 0 \Box R[1, 2] :\Box 0$. Further, to succinctly represent the program during the verification, numbers are marked before statements in programs. P_i^{jk} $(i, j \Box 1, 2, k \Box 1, \ldots 8)$ denotes the sub-program starting from line k in P_i^j to the end of P_i^j. For example, P_1^{12} implies the sub-program from line 2 in P_1^1 to the end of P_1^1, that is '`if` $(max[1] < hts[1, 2])$ `then` $\{max[1] :\Box hts[1, 2]\}$ `else` {skip} ; ... ;`if` $(rd[1, 2] \Box 1)$ `then` $\{rd[1, 2] :\Box 0$; `send`$(c_{12}, -1)\}$ `else` {skip}'.

Similarly, we can acquire PPTL properties for two processes. In the following, we only prove the first-come-first-served property, which is much more complex. Other properties can be proved in a similar manner. With PPTL, the FCFS property can be expressed as

$$A \equiv \Box(p_{12} \Box \Diamond(r_1 \Box \mathsf{skip} ; \Diamond r_2) \Box \neg p_{12} \Box \Diamond(r_2 \Box \mathsf{skip} ; \Diamond r_1))$$

For a concise representation, let $Q_1 \stackrel{\text{def}}{\Box} r_1 \Box \mathsf{skip} ; \Diamond r_2$, $Q_2 \stackrel{\text{def}}{\Box} r_2 \Box \mathsf{skip} ; \Diamond r_1$, $A_1 \stackrel{\text{def}}{\Box} p_{12} \Box \Diamond Q_1$, $A_2 \stackrel{\text{def}}{\Box} \neg p_{12} \Box \Diamond Q_2$. Then $A \equiv \Box(A_1 \Box A_2)$. Further, all the involved formulas, to which A is reduced, can be abbreviated as:

$A^1 \stackrel{\text{def}}{\Box} \Box(A_1 \Box A_2)$ $\quad$ $A^2 \stackrel{\text{def}}{\Box} \Diamond Q_2 \Box A^1$

$A^3 \stackrel{\text{def}}{\Box} \Diamond r_1 \Box A^1$ $\quad$ $A^4 \stackrel{\text{def}}{\Box} \Diamond Q_1 \Box A^1$

$A^5 \stackrel{\text{def}}{\Box} \Diamond r_2 \Box A^1$ $\quad$ $A^6 \stackrel{\text{def}}{\Box} \Diamond Q_2 \Box \Diamond r_1 \Box A^1$

$A^7 \stackrel{\text{def}}{\Box} \Diamond Q_2 \Box \Diamond Q_1 \Box A^1$ $\quad$ $A^8 \stackrel{\text{def}}{\Box} \Diamond Q_2 \Box \Diamond r_2 \Box A^1$

$A^9 \stackrel{\text{def}}{\Box} \Diamond r_1 \Box \Diamond Q_1 \Box A^1$ $\quad$ $A^{10} \stackrel{\text{def}}{\Box} \Diamond r_1 \Box \Diamond r_2 \Box A^1$

$A^{11} \stackrel{\text{def}}{\Box} \Diamond Q_1 \Box \Diamond r_2 \Box A^1$ $\quad$ $A^{12} \stackrel{\text{def}}{\Box} \Diamond Q_1 \Box \Diamond Q_2 \Box \Diamond r_1 \Box A^1$

$A^{13} \stackrel{\text{def}}{\Box} \Diamond Q_2 \Box \Diamond r_1 \Box \Diamond r_2 \Box A^1$ $\quad$ $A^{14} \stackrel{\text{def}}{\Box} \Diamond Q_1 \Box \Diamond Q_2 \Box \Diamond r_2 \Box A^1$

$A^{15} \stackrel{\text{def}}{\Box} \Diamond r_1 \Box \Diamond Q_1 \Box \Diamond r_2 \Box A^1$ $\quad$ $A^{16} \stackrel{\text{def}}{\Box} \Diamond Q_1 \Box \Diamond Q_2 \Box \Diamond r_1 \Box \Diamond r_2 \Box A^1$

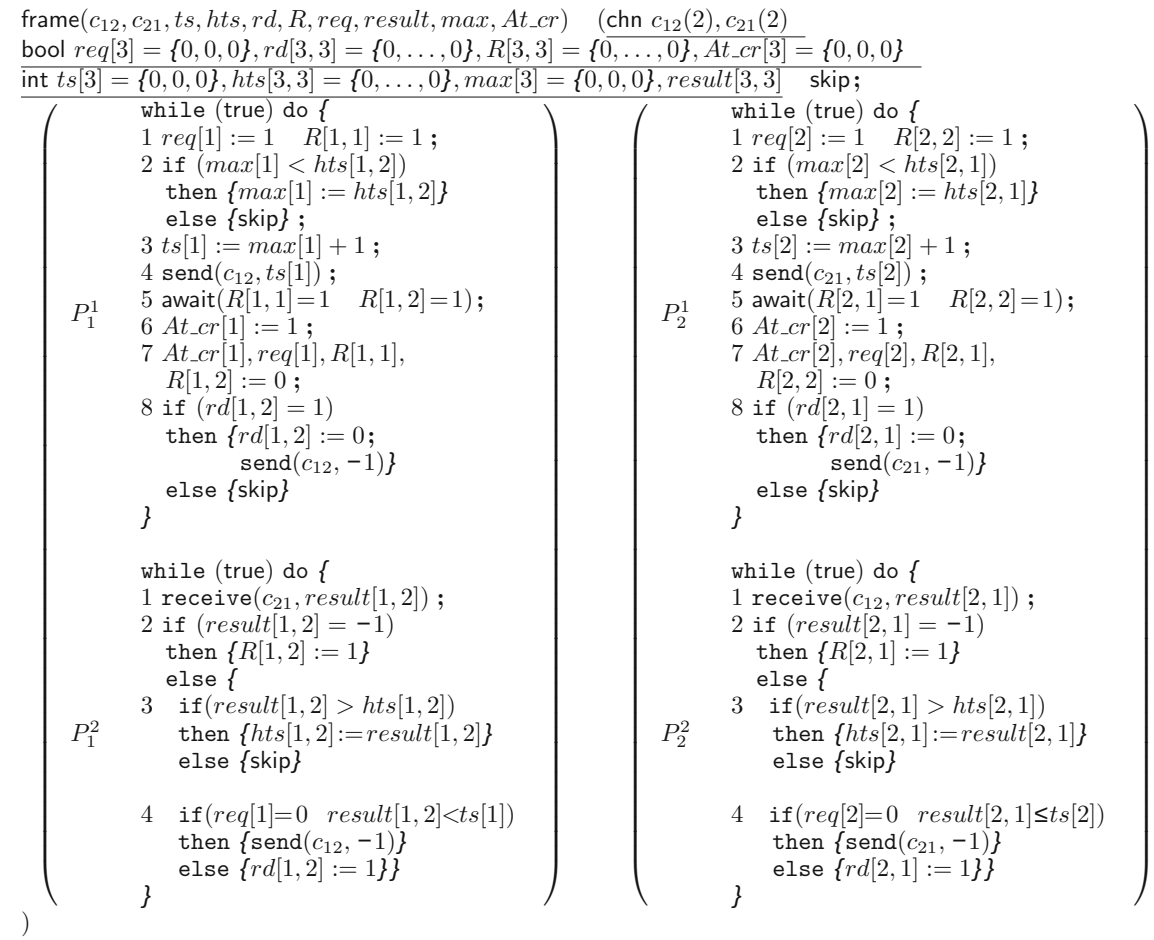

Fig. 3. The MSVL program for Ricart–Agrawala algorithm with two processes

Remember that we use P_f^i to denote the program at the state s_i with the present component P_c^i and the future component $\Box P_f^{i+1}$. Hence, the correctness assertion to verify is

$$\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \mathsf{true}\} \Box \{\sigma_0, A^1\}\ P_f^0\ \{\sigma_h, \mathsf{true}\} \tag{6}$$

where $h \Box \omega$. We firstly present Lemma 4.1 and 4.2, which specify some features about the program P_{RA2} and the FCFS property. Both the proofs of Lemma 4.1 and 4.2 proceed accompanying with the deduction of the correctness assertion and are given in the Appendix. In particular, Lemma 4.2 is proved on the basis of Lemma 4.1.

Lemma 4.1 *For RA program P_{RA2} in Fig. 3 and the FCFS property $\Box(A_1 \Box A_2)$:*

(1) *At state s_{24}, values of variables are as follows:* $max[1] \Box 1$, $max[2] \Box 2$, $hts[1, 2] \Box 3$, $hts[2, 1] \Box 2$, $ts[1] \Box 2$, $ts[2] \Box 3$, $req[1] \Box 0$, $req[2] \Box 1$, $rd[1, 2] \Box 0$, $rd[2, 1] \Box 0$, $At\_cr[1] \Box 0$, $At\_cr[2] \Box 0$, $R[1, 1] \Box 0$, $R[1, 2] \Box 0$, $R[2, 1] \Box 0$, $R[2, 2] \Box 1$, $c_{12} \Box \Box -1 \Box$, $c_{21} \Box \Box \Box$.

(2) *At state s_{14} and s_{25}, the program is* $P_f^{14} \overset{\text{def}}{\Box} \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((req[1] \Box 1 \Box R[1, 1] \Box 1 \Box \varepsilon ; P_1^{12} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (result[2, 1] \Box -1 \Box c_{12} \Box \Box \Box \Box \varepsilon ; P_2^{22} ; P_2^2))$.

(3) *At state s_{25}, the property is* $A^{12} \Box A^{14} \Box A^{16}$.

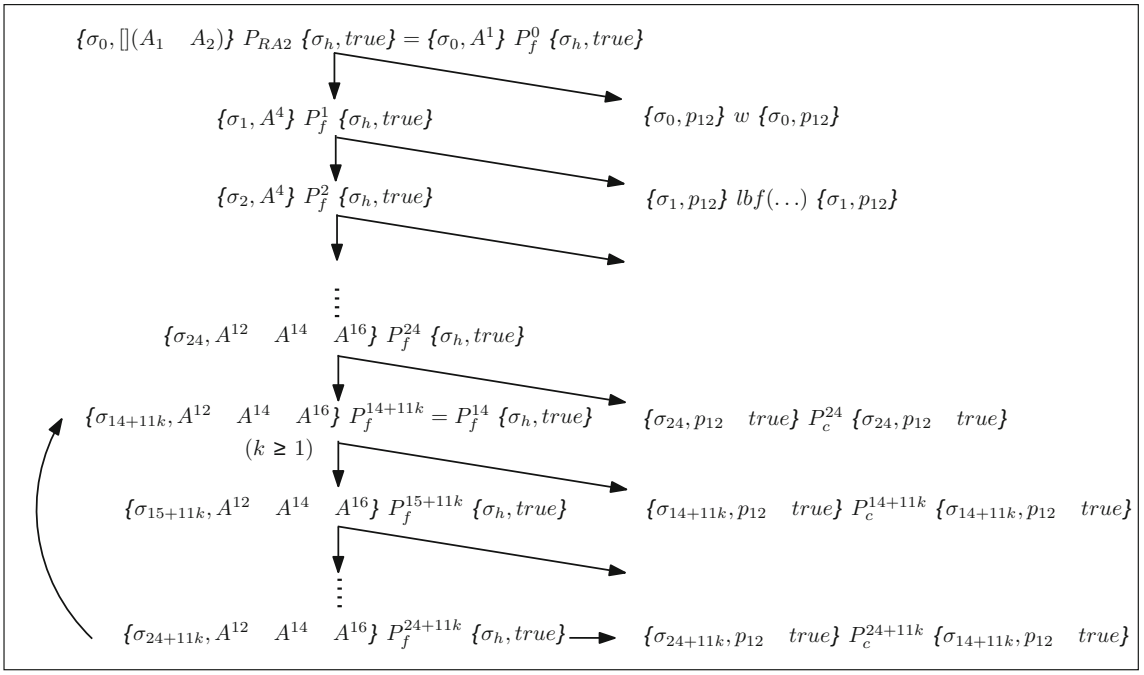

Fig. 4. The deduction process of the triple $\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \mathsf{true}\}$

Lemma 4.2 *For RA program P_{RA2} in Fig. 3 and the FCFS property $\Box(A_1 \Box A_2)$:*

(1) *At state $s_{13+11\Box k}$ $(k \geq 1)$, values of variables are as follows: $max[1] \Box 2k-1$, $max[2] \Box 2k$, $hts[1,2] \Box 2k+1$, $hts[2,1] \Box 2k$, $ts[1] \Box 2k$, $ts[2] \Box 2k+1$, $req[1] \Box 0$, $req[2] \Box 1$, $rd[1,2] \Box 0$, $rd[2,1] \Box 0$, $At\_cr[1] \Box 0$, $At\_cr[2] \Box 0$, $R[1,1] \Box 0$, $R[1,2] \Box 0$, $R[2,1] \Box 0$, $R[2,2] \Box 1$, $c_{12} \Box -1$, $c_{21} \Box$.*

(2) *At state $s_{14+11\Box k}$ $(k \geq 0)$, the program is $P_f^{14} \overset{\text{def}}{\Box} \mathsf{lbf}(\ldots)\Box\mathsf{frame}(\ldots)\Box((req[1] \Box 1\Box R[1,1] \Box 1\Box\varepsilon\ ;\ P_1^{12}\ ;\ P_1^1) \Box (P_1^{21}\ ;\ P_1^2) \Box (P_2^{15}\ ;\ P_2^1) \Box (result[2,1] \Box -1 \Box c_{12} \Box \Box \Box\ \varepsilon\ ;\ P_2^{22}\ ;\ P_2^2))$.*

(3) *At state $s_{14+11\Box k}$ $(k \geq 1)$, the property is $A^{12} \Box A^{14} \Box A^{16}$.*

From the proof of Lemma 4.1, we can find that for the program P_{RA2} and the property $\Box(A_1 \Box A_2)$, the present component of program can satisfy the present component of property at each state from state s_0 to s_{24}. Further, according to the proof of Lemma 4.2, we can also see that the present component of program satisfies the present component of property at each state from state s_{14+11k} $(k \geq 1)$ to $s_{13+11\Box(k+1)}$. Combining the proofs of Lemma 4.1 and Lemma 4.2, we can intuitively depict the deduction process of the triple $\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \mathsf{true}\}$ in Fig. 4. Thereinto, each triple $\{\sigma_i, A\}\ P_f^i\ \{\sigma_h, true\}$ at the beginning of the arrows can be decomposed into two triples that the arrows point to, that is $\{\sigma_i, A_c\}\ P_c^i\ \{\sigma_i, A_c\}$ and $\{\sigma_{i+1}, A_{i+1}\}\ P_f^{i+1}\ \{\sigma_h, \mathsf{true}\}$, where $P_f^i \Box P_c^i\Box\Box P_f^{i+1}$ and $A \equiv A_c \Box \Box A_{i+1}$. The former holds if present component P_c^i of P_f^i satisfies the present component A_c of A by axiom (APC) while the latter can be further decomposed at the next state s_{i+1} over an interval σ_h. In this way, triples in the form of $\{\sigma_i, A_c\}\ P_c^i\ \{\sigma_i, A_c\}$ for every state over the interval σ_h are generated. Hence, to prove $\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \mathsf{true}\}$ holds is to prove at each state the triple $\{\sigma_i, A_c\}\ P_c^i\ \{\sigma_i, A_c\}$ holds.

Theorem 4.1 *The instance of RA algorithm with two processes satisfies the first-come-first-served property.*

Proof. To prove $\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \mathsf{true}\}$ holds is to prove at each state the triple $\{\sigma_i, A_c\}\ P_c^i\ \{\sigma_i, A_c\}$ holds. By axiom (APC), this is exactly to prove the present component of program can satisfy the present component of property at each state over the interval σ_h. Firstly, by Lemma 4.1, we know that the present component of property is satisfied by the present component of program at each state from state s_0 to state s_{24}. Further, in accordance to the proof of Lemma 4.2, it is easy to obtain that at each state from state $s_{14+11\Box k}$ $(k \geq 1)$ to state $s_{13+11\Box(k+1)}$, the present component of property can be satisfied by the present component of program. Therefore,

we can conclude that the present component of program satisfies the present component of property at each state over interval σ_h, which results in $\{\sigma_0, \Box(A_1 \Box A_2)\}\ P_{RA2}\ \{\sigma_h, \text{true}\}$ holds. Hence, the instance of RA algorithm satisfies the first-come-first-served property. □

5. Related work

Due to the complexity and significance of distributed systems, numerous formalisms [Bru96, Mil82, Hoa78, Mil99, Hen07, LT87, RNP13, Pet77, Tan83] are put forward to guarantee their correctness. Within process algebra community, CCS [Bru96, Mil82] and CSP [Hoa78] have been widely used to specify and verify distributed systems owing to their effective mechanisms of message passing. CCS is based on asynchronous communication while CSP is on synchronous communication. Following them, in order to conveniently support various data types in distributed systems, μCRL and mCRL2 [CGK+13] are proposed. Further, on the basis of CCS, Pi-calculus [Mil99] allows channel names to be communicated along the channels themselves, which facilitates describing mobility in distributed systems. Whereafter, distributed Pi-calculus [Hen07] enriches Pi-calculus with a network layer, a primitive migration construct and types.

With the development of automata, variants of automata, such as I/O automata [LT87], timed automata [RNP13], can be applied to the verification of distributed systems. An I/O automaton [LT87] associates transitions with named actions, i.e. input, output and internal actions. Input and output actions can express receiving and sending messages between processes while internal actions can describe interior behaviors of processes and channels. In this way, an individual component in a distributed system, namely a process or a message channel, is modeled as an I/O automation and then the whole distributed system can be modeled as the composition of many I/O automata. Timed automata [RNP13] specially aim at the distributed real-time systems and can capture different sorts of relationships among computer clocks of a distributed system.

Further, Petri nets [Pet77] and its family, like colored Petri nets [Jen91], also make a great contribution to this area. Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution, which can be understood clearly and intuitively. Meanwhile, Petri nets also have an exactly mathematical semantics definition with a well-developed mathematical theory for process analysis.

Compared with these formal modeling languages, MSVL is a different formalism of an interval based temporal logic programming language and has three advantages: (1) Since MSVL is an executable subset of PTL, we do not need to simultaneously use two separate notations during verification. This means that writing programs, specifying their properties and verifying those properties, can all be treated within the same notation, which delicately avoids complex transformations between different notations; (2) The formalisms of process algebra, automata and Petri nets provide tools for the high-level description and are for specifying and verifying distributed systems rather than for programming. In contrast, MSVL offers not only an abstract model, but also a practical programming language. Particularly, MSVL is capable of specifying a system at any level of abstraction–from the highest-level specification to the programming language implementation. Thus, hierarchical design methods of distributed systems can be directly supported in MSVL without extra mechanisms to link the different levels of description, which further facilitates hierarchical specification and reasoning about distributed systems; (3) MSVL has certain characteristics in common with imperative programming, such as assignment statement '$x \Box e$', conditional statement '`if-then-else`', iterative statement '`while-do`'. To some degree, these constructs bridge the gap between temporal logic programming and imperative programming in the sense that they enable users, who are familiar with imperative programming, to learn MSVL relatively more easily and write temporal logic programs in a structured style.

Most of temporal logic programming languages aforementioned in Sect. 1 lack constructs to express asynchronous communication and channels, which further results in they fail to verify distributed systems. XYZ/E [Tan83] and concurrent METATEM [Fis94] are two languages that can be used in modeling and verifying distributed systems. For XYZ/E, a channel is defined as a channel variable which can be a parameter of a process. This method is flexible, but conflicts might happen when more than one process accesses the same channel at the same time. A couple of Hoare-style rules for XYZ/E is proposed but only applicable to a structured version of XYZ/E. Contrarily, in MSVL, a channel is unidirectional and only connects two processes, which elegantly avoids such conflicts. The axiomatic system of MSVL can deal with all the statements within MSVL. Based on METATEM, concurrent METATEM is mainly for distributed artificial systems and its communication mechanism is through a broadcast message-passing. To the best of our knowledge, there are no axiomatic systems in terms of a Hoare logic-like triple for concurrent METATEM.

Besides the formalisms above, there also exist other approaches, which differ from ours in the underlying paradigms. In [CY83], an event-based model is developed to formally specify the behavior and the structure of distributed systems. Furthermore, Event-B [D13] models and analyzes distributed systems at the system level as well as ensures the correctness of distributed system by means of refinement and consistency checking.

In [MWD11], the authors extended MSVL with asynchronous communication commands and channels to specify message exchanging in distributed systems. It also verified an instance of three processes of a distributed contract signing protocol through model checking. However, since the model checking technique suffers from the state explosion problem, their method fails in the verification of infinite state systems, e.g. RA algorithm. In the contrary, we concerns the verification of distributed systems with MSVL by means of theorem proving, which can handle both finite state and infinite state systems. To this end, we extends the axiomatic system of MSVL with some state axioms regarding asynchronous communication commands.

6. Conclusion

In order to verify distributed systems with MSVL in a deductive way, this paper enriched the axiomatic system of MSVL in [YDM10] with axioms for asynchronous communication. To do this, firstly state axioms regarding asynchronous communication commands were formalized and their soundness and completeness were proved. Secondly, to show how to use the extended axiomatic system, RA algorithm was taken into account. In particular, its model, properties and a minutely verifying procedure with respect to the first-come-first-served property were presented. As we can see, our method indeed can verify distributed systems with message passing in a suitable and natural manner.

In this paper, although we only demonstrate the verification for two processes of RA algorithm, other instances can be proved similarly. However, the parameterized verification for arbitrarily N processes is still a challenge to us at present. Thus, in a near future, we will investigate the approach for parameterized verification with the extended axiomatic system of MSVL. Moreover, we will develop an assistant tool to facilitate using our technique mechanically. Besides, we will explore more case studies for practicability.

Acknowledgements

We are grateful to the anonymous reviewers for their valuable comments and suggestions. This work is supported by the National Natural Science Foundation of China [grant numbers 61133001, 61272118, 61272117, 61202038, 61322202, 91218301]; and National Program on Key Basic Research Project (973 Program) [grant number 2010CB328102].

Appendix

Proof of Lemma 3.1:

Proof. We demonstrate it in terms of the form of P and the logic law (L6) is used by default:

(1) P is $\mathtt{send}(c, x)$:

$$
\begin{aligned}
& \mathtt{send}(c, x) \\
\equiv\ & \mathsf{await}(\neg \mathit{isfull}(c)) \,;\, c :\Box\ c \cdot \Box x\Box \\
\equiv\ & (\neg \mathit{isfull}(c) \wedge \varepsilon) \vee \mathit{isfull}(c) \wedge \bigcirc(\mathsf{lbf}(c) \wedge \mathsf{await}(\neg \mathit{isfull}(c)))) \,;\, c :\Box\ c \cdot \Box x\Box && \text{(L5)} \\
\equiv\ & \neg \mathit{isfull}(c) \wedge \varepsilon \,;\, c :\Box\ c \cdot \Box x\Box \vee \mathit{isfull}(c) \wedge \bigcirc(\mathsf{lbf}(c) \wedge \mathsf{await}(\neg \mathit{isfull}(c))) \,;\, c :\Box\ c \cdot \Box x\Box && \text{(L4)} \\
\equiv\ & \neg \mathit{isfull}(c) \wedge c :\Box\ c \cdot \Box x\Box \vee \mathit{isfull}(c) \wedge \bigcirc(\mathsf{lbf}(c) \wedge \mathsf{await}(\neg \mathit{isfull}(c)) \,;\, c :\Box\ c \cdot \Box x\Box) && \text{(L1–L3)} \\
\equiv\ & \neg \mathit{isfull}(c) \wedge \bigcirc(c \wedge c^{\Box} \wedge \varepsilon) \vee \mathit{isfull}(c) \wedge \bigcirc(\mathsf{lbf}(c) \wedge \mathtt{send}(c, x)) && \text{(L2)}
\end{aligned}
$$

where $c^{\Box} \wedge c \cdot \Box x\Box$ is a constant. If the predicate '$\mathit{isfull}(c)$' is true, the normal form of $\mathtt{send}(c, x)$ is $\bigcirc(\mathsf{lbf}(c) \wedge \mathtt{send}(c, x))$; otherwise, its normal form is $\bigcirc(c \wedge c^{\Box} \wedge \varepsilon)$.

(2) P is $\texttt{receive}(c, x)$:

$$
\begin{array}{ll}
& \texttt{receive}(c, x) \\
\equiv & \mathsf{await}(\neg \mathit{isempty}(c)) \,;\, x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c) \\
\equiv & (\neg \mathit{isempty}(c) \Box\ \varepsilon \Box\ \mathit{isempty}(c) \Box\ \Box(\mathsf{lbf}(c) \Box\ \mathsf{await}(\neg \mathit{isempty}(c)))) ; x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c) \quad \text{(L5)} \\
\equiv & \neg \mathit{isempty}(c) \Box\ x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c) \Box \\
& \mathit{isempty}(c) \Box\ \Box(\mathsf{lbf}(c) \Box\ \mathsf{await}(\neg \mathit{isempty}(c)) \,;\, x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c)) \quad \text{(L1–L4)} \\
\equiv & \neg \mathit{isempty}(c) \Box\ \Box(x \Box\ \mathit{head}(c) \Box\ c \Box\ c^{\frown} \Box\ \varepsilon) \Box\ \mathit{isempty}(c) \Box\ \Box(\mathsf{lbf}(c) \Box\ \texttt{receive}(c, x)) \quad \text{(L2)}
\end{array}
$$

where $c^{\frown} \Box\ \mathit{tail}(c)$ is a constant. If the predicate '$\mathit{isempty}(c)$' is true, the normal form of $\texttt{receive}(c, x)$ is $\Box(\mathsf{lbf}(c) \Box\ \texttt{receive}(c, x))$; or else, its normal form is $\Box(x \Box\ \mathit{head}(c) \Box\ c \Box\ c^{\frown} \Box\ \varepsilon)$.

(3) P is $\texttt{put}(c, x)$:

$$
\begin{array}{ll}
& \texttt{put}(c, x) \\
\equiv & \texttt{if}\ (\neg \mathit{isfull}(c))\ \texttt{then}\ \{c :\Box\ c \cdot \lceil x \rceil\}\ \texttt{else}\ \{\mathsf{skip}\} \quad \text{(definition of put)} \\
\equiv & \neg \mathit{isfull}(c) \Box\ c :\Box\ c \cdot \lceil x \rceil \Box\ \mathit{isfull}(c) \Box\ \mathsf{skip} \quad \text{(definition of if)} \\
\equiv & \neg \mathit{isfull}(c) \Box\ \Box(c \Box\ c^{\frown} \Box\ \varepsilon) \Box\ \mathit{isfull}(c) \Box\ \Box\varepsilon \quad \text{(definition of } :\Box\text{)}
\end{array}
$$

where $c^{\frown} \Box\ c \cdot \lceil x \rceil$ is a constant. If the predicate '$\mathit{isfull}(c)$' is true, $\texttt{put}(c, x)$ can be reduced into $\Box\varepsilon$; otherwise, it can be reduced into $\Box(c \Box\ c^{\frown} \Box\ \varepsilon)$.

(4) P is $\texttt{get}(c, x)$:

$$
\begin{array}{ll}
& \texttt{get}(c, x) \\
\equiv & \texttt{if}\ (\neg \mathit{isempty}(c))\ \texttt{then}\ \{x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c)\}\ \texttt{else}\ \{\mathsf{skip}\} \quad \text{(definition of get)} \\
\equiv & \neg \mathit{isempty}(c) \Box\ x :\Box\ \mathit{head}(c) \Box\ c :\Box\ \mathit{tail}(c) \Box\ \mathit{isempty}(c) \Box\ \mathsf{skip} \quad \text{(definition of if)} \\
\equiv & \neg \mathit{isempty}(c) \Box\ \Box(x \Box\ \mathit{head}(c) \Box\ c \Box\ c^{\frown} \Box\ \varepsilon) \Box\ \mathit{isempty}(c) \Box\ \Box\varepsilon \quad \text{(definition of } :\Box\text{)}
\end{array}
$$

where $c^{\frown} \Box\ \mathit{tail}(c)$ is a constant. If the predicate '$\mathit{isempty}(c)$' is true, $\texttt{get}(c, x)$ can be reduced into $\Box\varepsilon$; or else, it can be reduced into $\Box(x \Box\ \mathit{head}(c) \Box\ c \Box\ c^{\frown} \Box\ \varepsilon)$. □

Proof of Lemma 4.1:

Proof. We will prove the lemma accompanying with the deduction of the correctness assertion. For brevity, we omit the minutely step-by-step transformation from program and property to their normal forms but only give the results. At state s_0, the normal form of property $\Box(A_1 \Box\ A_2) \equiv A^1$ is

$$A^1 \equiv \underline{p_{12} \Box\ \Box A^4} \Box\ p_{12} \Box\ r_1 \Box\ \Box A^5 \Box\ \neg p_{12} \Box\ \Box A^2 \Box\ \neg p_{12} \Box\ r_2 \Box\ \Box A^3$$

and by state axioms in Tables 4 and 6, the normal form of program P_{RA} is

$$P_{RA2} \boxplus P_f^0 \boxplus w \Box\ \Box P_f^1 \boxplus w \Box\ \Box(\mathsf{lbf}(\ldots) \Box\ \mathsf{frame}(\ldots) \Box\ (P_1^1 \Box\ P_1^2 \Box\ P_2^1 \Box\ P_2^2))$$

As in w, both $ts[1]$ and $ts[2]$ are equal to 0, it is obvious that $w \rightarrow ts[1] \leq ts[2]$(i.e. p_{12}). Thus by rule (ANext) we have:

$$\{\sigma_0, A^1\}\ P_f^0\ \{\sigma_h, \mathsf{true}\}\ \Box\Box\ \{\sigma_0, p_{12}\}\ w\ \{\sigma_0, p_{12}\} \text{ and } \{\sigma_1, A^4\}\ P_f^1\ \{\sigma_h, \mathsf{true}\}$$

According to axiom (APC) in Table 4, it is easy to see that $\{\sigma_0, p_{12}\}\ w\ \{\sigma_0, p_{12}\}$ holds, that is the present component of program satisfies the present component of property at state s_0. Therefore, in the following, we need to prove the triple $\{\sigma_1, A^4\}\ P_f^1\ \{\sigma_h, \mathsf{true}\}$ holds. In this way, the deduction proceeds state-by-state and on-the-fly. Rules (APC) and (ANext) can be utilized in an analogous manner. Due to the limited space, in what to follow, we briefly present the deduction process.

At state s_1, the program is $P_f^1 \boxplus \mathsf{lbf}(\ldots) \Box\ \Box P_f^2$,

$$\{\sigma_1, A^4\}\ P_f^1\ \{\sigma_h, \mathsf{true}\}\ \Box\Box\ \{\sigma_1, p_{12}\}\ \mathsf{lbf}(\ldots)\ \{\sigma_1, p_{12}\} \text{ and } \{\sigma_2, A^4\}\ P_f^2\ \{\sigma_h, \mathsf{true}\}$$

Since the state formula $\mathsf{lbf}(ts[1], ts[2])$ implies the values of $ts[1]$ and $ts[2]$ at state s_1 are the same as that at state s_0, so $ts[1] \Box\ 0$ and $ts[2] \Box\ 0$, which further leads to $\mathsf{lbf}(\ldots) \rightarrow p_{12}$. Hence, the present component of program satisfies the present component of property at state s_1.

At state s_2, the program is $P_f^2 \boxplus \mathsf{lbf}(\ldots) \Box\ \mathsf{frame}(\ldots) \Box\ ((req[1] \Box\ 1 \Box\ R[1, 1] \Box\ 1 \Box\ P_1^{12} \,;\, P_1^1) \Box\ (P_1^{21} \,;\, P_1^2) \Box\ (req[2] \Box\ 1 \Box\ R[2, 2] \Box\ 1 \Box\ P_2^{12} \,;\, P_2^1) \Box\ (P_2^{21} \,;\, P_2^2))$:

$\{\sigma_2, A^4\}\ P_f^2\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_2, p_{12}\}\ P_c^2\ \{\sigma_2, p_{12}\}$ and $\{\sigma_3, A^4\}\ P_f^3\ \{\sigma_h, \mathsf{true}\}$

where $P_c^2 \triangleq req[1] \square 1 \square R[1,1] \square 1 \square req[2] \square 1 \square R[2,2] \square 1 \square \mathsf{lbf}(\ldots)$. Since $ts[1] \square 0 \square ts[2] \square 0$, obviously $P_c^2 \rightarrow p_{12}$. Thus, the present component of program satisfies the present component of property at state s_2.

At state s_3, the program is $P_f^3 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{13}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (P_2^{13}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_3, A^4\}\ P_f^3\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_3, p_{12}\}\ P_c^3\ \{\sigma_3, p_{12}\}$ and $\{\sigma_4, A^4\}\ P_f^4\ \{\sigma_h, \mathsf{true}\}$

where $P_c^3 \triangleq \mathsf{lbf}(\ldots)$. Clearly, $ts[1] \square 0 \square ts[2] \square 0$ results in $P_c^3 \rightarrow p_{12}$.

At state s_4, the program is $P_f^4 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((ts[1] \square 1 \square \varepsilon\ ;P_1^{14}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (ts[2] \square 1 \square \varepsilon\ ;P_2^{14}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_4, A^4\}\ P_f^4\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_4, p_{12}\}\ P_c^4\ \{\sigma_4, p_{12}\}$ and $\{\sigma_5, A^4\}\ P_f^5\ \{\sigma_h, \mathsf{true}\}$

where $P_c^4 \triangleq ts[1] \square 1 \square ts[2] \square 1 \square \mathsf{lbf}(\ldots)$. It is obvious that $P_c^4 \rightarrow p_{12}$ holds.

At state s_5, the program is $P_f^5 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((c_{12} \square \square 1 \square \square \varepsilon\ ;P_1^{15}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (c_{21} \square \square 1 \square \square \varepsilon\ ;P_2^{15}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_5, A^4\}\ P_f^5\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_5, p_{12}\}\ P_c^5\ \{\sigma_5, p_{12}\}$ and $\{\sigma_6, A^4\}\ P_f^6\ \{\sigma_h, \mathsf{true}\}$

where $P_c^5 \triangleq c_{12} \square \square 1 \square \square c_{21} \square \square 1 \square \square \mathsf{lbf}(\ldots)$. Thus, $ts[1] \square 1 \square ts[2] \square 1$, which leads to $P_c^5 \rightarrow p_{12}$.

At state s_6, the program is $P_f^6 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (result[1,2] \square 1 \square c_{21} \square \square \square \varepsilon\ ;P_1^{22}\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (result[2,1] \square 1 \square c_{12} \square \square \square \varepsilon\ ;P_2^{22}\ ;P_2^2))$:

$\{\sigma_6, A^4\}\ P_f^6\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_6, p_{12}\}\ P_c^6\ \{\sigma_6, p_{12}\}$ and $\{\sigma_7, A^4\}\ P_f^7\ \{\sigma_h, \mathsf{true}\}$

where $P_c^6 \triangleq result[1,2] \square 1 \square c_{21} \square \square \square result[2,1] \square 1 \square c_{12} \square \square \square \mathsf{lbf}(\ldots)$. Hence, we can obtain $ts[1] \square 0 \square ts[2] \square 0$ and $P_c^6 \rightarrow p_{12}$, which indicates the present component of program satisfies the present component of property at state s_6.

At state s_7, the program is $P_f^7 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (hts[1,2] \square 1 \square rd[1,2] \square 1 \square \varepsilon\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (hts[2,1] \square 1 \square c_{21} \square \square -1 \square \square \varepsilon\ ;P_2^2))$:

$\{\sigma_7, A^4\}\ P_f^7\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_7, p_{12}\}\ P_c^7\ \{\sigma_7, p_{12}\}$ and $\{\sigma_8, A^4\}\ P_f^8\ \{\sigma_h, \mathsf{true}\}$

where $P_c^7 \triangleq hts[1,2] \square 1 \square rd[1,2] \square 1 \square hts[2,1] \square 1 \square c_{21} \square \square -1 \square \square \mathsf{lbf}(\ldots)$. As $ts[1] \square 1 \square ts[2] \square 1$, clearly $P_c^7 \rightarrow p_{12}$.

At state s_8, the program is $P_f^8 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (result[1,2] \square -1 \square c_{21} \square \square \square \varepsilon\ ;P_1^{22}\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_8, A^4\}\ P_f^8\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_8, p_{12}\}\ P_c^8\ \{\sigma_8, p_{12}\}$ and $\{\sigma_9, A^4\}\ P_f^9\ \{\sigma_h, \mathsf{true}\}$

where $P_c^8 \triangleq result[1,2] \square -1 \square c_{21} \square \square \square \mathsf{lbf}(\ldots)$. It is obvious $ts[1] \square 1 \square ts[2] \square 1$, which results in $P_c^8 \rightarrow p_{12}$.

At state s_9, the program is $P_f^9 \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (R[1,2] \square 1 \square \varepsilon\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_9, A^4\}\ P_f^9\ \{\sigma_h, \mathsf{true}\}\ \square\square\ \{\sigma_9, p_{12}\}\ P_c^9\ \{\sigma_9, p_{12}\}$ and $\{\sigma_{10}, A^4\}\ P_f^{10}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^9 \triangleq R[1,2] \square 1 \square \mathsf{lbf}(\ldots)$. It is clear $ts[1] \square 1 \square ts[2] \square 1$ and $P_c^9 \rightarrow p_{12}$ holds.

At state s_{10}, the program is $P_f^{10} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((At\_cr[1] \square 1 \square \varepsilon\ ;P_1^{17}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$\{\sigma_{10}, A^4\}\ P_f^{10}\ \{\sigma_h, \mathsf{true}\}\ \square\square$
$\{\sigma_{10}, p_{12}\}\ P_c^{10}\ \{\sigma_{10}, p_{12}\}$ and $\{\sigma_{11}, A^4\}\ P_f^{11}\ \{\sigma_h, \mathsf{true}\}$
or
$\{\sigma_{10}, p_{12} \square r_1\}\ P_c^{10}\ \{\sigma_{10}, p_{12}\}$ and $\{\sigma_{11}, A^5\}\ P_f^{11}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{10} \triangleq At\_cr[1] \square 1 \square \mathsf{lbf}(\ldots)$. Since in P_c^{10}, $ts[1] \square 1$, $ts[2] \square 1$, $At\_cr[1] \square 1$, P_c^{10} can infer p_{12} and $p_{12} \square r_1$, which further leads to that two distinct branches can be deduced from the next state s_{11}. Then we can arbitrarily select one of them to deduce and the other can be deduced in an analogous fashion. In the following, we give the deduction for $\{\sigma_{11}, A^5\}P_f^{11}\{\sigma_h, \mathsf{true}\}$.

At state s_{11}, the program is $P_f^{11} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((At\_cr[1] \Box 0 \Box req[1] \Box 0 \Box R[1,1] \Box 0 \Box R[1,2] \Box 0 \Box \varepsilon ; P_1^{18} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (P_2^{21} ; P_2^2))$:

$$\{\sigma_{11}, A^5\}\ P_f^{11}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{11}, p_{12}\}\ P_c^{11}\ \{\sigma_{11}, p_{12}\} \text{ and } \{\sigma_{12}, A^{11}\}\ P_f^{12}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{11} \triangleq At\_cr[1] \Box 0 \Box req[1] \Box 0 \Box R[1,1] \Box 0 \Box R[1,2] \Box 0 \Box \mathsf{lbf}(\ldots)$. Thus, we can obtain $ts[1] \Box 1$, $ts[2] \Box 1$. Therefore obviously $P_c^{11} \rightarrow p_{12}$ holds.

At state s_{12}, the program is $P_f^{12} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((rd[1,2] \Box 0 \Box \varepsilon ; \mathtt{send}(c_{12}, -1) ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (P_2^{21} ; P_2^2))$:

$$\{\sigma_{12}, A^{11}\}\ P_f^{12}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{12}, p_{12}\}\ P_c^{12}\ \{\sigma_{12}, p_{12}\} \text{ and } \{\sigma_{13}, A^{11}\}\ P_f^{13}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{12} \triangleq rd[1,2] \Box 0 \Box \mathsf{lbf}(\ldots)$. Clearly $ts[1] \Box 1 \Box ts[2] \Box 1$ results in $P_c^{12} \rightarrow p_{12}$.

At state s_{13}, the program is $P_f^{13} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((c_{12} \Box \Box -1 \Box \Box \varepsilon ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (P_2^{21} ; P_2^2))$:

$$\{\sigma_{13}, A^{11}\}\ P_f^{13}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{13}, p_{12}\}\ P_c^{13}\ \{\sigma_{13}, p_{12}\} \text{ and } \{\sigma_{14}, A^{11}\}\ P_f^{14}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{13} \triangleq c_{12} \Box \Box -1 \Box \Box \mathsf{lbf}(\ldots)$. Thus $ts[1] \Box 1 \Box ts[2] \Box 1$ and $P_c^{13} \rightarrow p_{12}$ holds.

At state s_{14}, the program is $P_f^{14} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((req[1] \Box 1 \Box R[1,1] \Box 1 \Box \varepsilon ; P_1^{12} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (result[2,1] \Box -1 \Box c_{12} \Box \Box\Box \Box \varepsilon ; P_2^{22} ; P_2^2))$:

$$\{\sigma_{14}, A^{11}\}\ P_f^{14}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{14}, p_{12}\}\ P_c^{14}\ \{\sigma_{14}, p_{12}\} \text{ and } \{\sigma_{15}, A^{11}\}\ P_f^{15}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{14} \triangleq req[1] \Box 1 \Box R[1,1] \Box 1 \Box result[2,1] \Box -1 \Box c_{12} \Box \Box\Box \Box \mathsf{lbf}(\ldots)$. It is obvious $ts[1] \Box 1 \Box ts[2] \Box 1$ and $P_c^{14} \rightarrow p_{12}$.

At state s_{15}, the program is $P_f^{15} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((max[1] \Box 1 \Box \varepsilon ; P_1^{13} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{15} ; P_2^1) \Box (R[2,1] \Box 1 \Box \varepsilon ; P_2^2))$:

$$\{\sigma_{15}, A^{11}\}\ P_f^{15}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{15}, p_{12}\}\ P_c^{15}\ \{\sigma_{15}, p_{12}\} \text{ and } \{\sigma_{16}, A^{11}\}\ P_f^{16}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{15} \triangleq R[2,1] \Box 1 \Box max[1] \Box 1 \Box \mathsf{lbf}(\ldots)$. Therefore obviously we have $ts[1] \Box 1 \Box ts[2] \Box 1$ and $P_c^{15} \rightarrow p_{12}$.

At state s_{16}, the program is $P_f^{16} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((ts[1] \Box 2 \Box \varepsilon ; P_1^{14} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (At\_cr[2] \Box 1 \Box \varepsilon ; P_2^{17} ; P_2^1) \Box (P_2^{21} ; P_2^2))$:

$$\{\sigma_{16}, A^{11}\}\ P_f^{16}\ \{\sigma_h, \mathsf{true}\} \Box\Box \begin{array}{l} \{\sigma_{16}, \neg p_{12}\}\ P_c^{16}\ \{\sigma_{16}, p_{12}\} \text{ and } \{\sigma_{17}, A^{14}\}\ P_f^{17}\ \{\sigma_h, \mathsf{true}\} \\ \text{or} \\ \{\sigma_{16}, \neg p_{12} \Box r_2\} P_c^{16} \{\sigma_{16}, \neg p_{12} \Box r_2\} \text{ and } \{\sigma_{17}, A^{15} \Box A^9 \Box A^7\} P_f^{17} \{\sigma_h, \mathsf{true}\} \end{array}$$

where $P_c^{16} \triangleq ts[1] \Box 2 \Box At\_cr[2] \Box 1 \Box \mathsf{lbf}(\ldots)$. Since $ts[1] \Box 2$, $ts[1] \Box 1$ and $At\_cr[2] \Box 1$, P_c^{16} can imply both $\neg p_{12}$ and $\neg p_{12} \Box r_2$. Similarly, we choose the triple $\{\sigma_{17}, A^{14}\}\ P_f^{17}\ \{\sigma_h, \mathsf{true}\}$ to deduce at the state s_{17}.

At state s_{17}, the program is $P_f^{17} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((c_{12} \Box \Box 2 \Box \Box \varepsilon ; P_1^{15} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (At\_cr[2] \Box 0 \Box req[2] \Box 0 \Box R[2,1] \Box 0 \Box R[2,2] \Box 0 \Box \varepsilon ; P_2^{18} ; P_2^1) \Box (P_2^{21} ; P_2^2))$:

$$\{\sigma_{17}, A^{14}\}\ P_f^{17}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{17}, \mathsf{true}\}\ P_c^{17}\ \{\sigma_{17}, \mathsf{true}\} \text{ and } \{\sigma_{18}, A^{14}\}\ P_f^{18}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{17} \triangleq c_{12} \Box \Box 2 \Box \Box At\_cr[2] \Box 0 \Box req[2] \Box 0 \Box R[2,1] \Box 0 \Box R[2,2] \Box 0 \Box \mathsf{lbf}(\ldots)$. Any formula can infer true.

At state s_{18}, the program is $P_f^{18} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((P_1^{15} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (P_2^{11} ; P_2^1) \Box (result[2,1] \Box 2 \Box c_{12} \Box \Box\Box \Box \varepsilon ; P_2^{22} ; P_2^2))$:

$$\{\sigma_{18}, A^{14}\}\ P_f^{18}\ \{\sigma_h, \mathsf{true}\} \Box\Box \{\sigma_{18}, \mathsf{true}\}\ P_c^{18}\ \{\sigma_{18}, \mathsf{true}\} \text{ and } \{\sigma_{19}, A^{14}\}\ P_f^{19}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{18} \triangleq result[2,1] \Box 2 \Box c_{12} \Box \Box\Box \Box \mathsf{lbf}(\ldots)$. Obviously $P_c^{18} \rightarrow \mathsf{true}$.

At state s_{19}, the program is $P_f^{19} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((P_1^{15} ; P_1^1) \Box (P_1^{21} ; P_1^2) \Box (req[2] \Box 1 \Box R[2,2] \Box 1 \Box \varepsilon ; P_2^{12} ; P_2^1) \Box (hts[2,1] \Box 2 \Box c_{21} \Box \Box -1 \Box \Box \varepsilon ; P_2^2))$:

$\{\sigma_{19}, A^{14}\}\ P_f^{19}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{19}, \mathsf{true}\}\ P_c^{19}\ \{\sigma_{19}, \mathsf{true}\}$ and $\{\sigma_{20}, A^{14}\}\ P_f^{20}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{19} \triangleq req[2] \square 1 \square R[2,2] \square 1 \square hts[2,1] \square 2 \square c_{21} \square \square -1 \square \square \mathsf{lbf}(\ldots)$. Clearly $P_c^{19} \rightarrow \mathsf{true}$.

At state s_{20}, the program is $P_f^{20} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}; P_1^1) \square (result[1,2] \square -1 \square c_{21} \square \square \square \varepsilon; P_1^{22}; P_1^2) \square (max[2] \square 2 \square \varepsilon; P_2^{13}; P_2^1) \square (P_2^{21}; P_2^2))$:

$\{\sigma_{20}, A^{14}\}\ P_f^{20}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{20}, \mathsf{true}\}\ P_c^{20}\ \{\sigma_{20}, \mathsf{true}\}$ and $\{\sigma_{21}, A^{14}\}\ P_f^{21}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{20} \triangleq result[1,2] \square -1 \square c_{21} \square \square \square max[2] \square 2 \square \mathsf{lbf}(\ldots)$. It is clear $P_c^{20} \rightarrow \mathsf{true}$.

At state s_{21}, the program is $P_f^{21} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}; P_1^1) \square (R[1,2] \square 1 \square \varepsilon; P_1^2) \square (ts[2] \square 3 \square \varepsilon; P_2^{14}; P_2^1) \square (P_2^{21}; P_2^2))$:

$\{\sigma_{21}, A^{14}\}\ P_f^{21}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{21}, \mathsf{true}\}\ P_c^{21}\ \{\sigma_{21}, \mathsf{true}\}$ and $\{\sigma_{22}, A^{14}\}\ P_f^{22}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{21} \triangleq R[1,2] \square 1 \square ts[2] \square 3 \square \mathsf{lbf}(\ldots)$. It is obvious $P_c^{21} \rightarrow \mathsf{true}$.

At state s_{22}, the program is $P_f^{22} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((At\_cr[1] \square 1 \square \varepsilon; P_1^{17}; P_1^1) \square (P_1^{21}; P_1^2) \square (c_{21} \square \square 3 \square \square \varepsilon; P_2^{15}; P_2^1) \square (P_2^{21}; P_2^2))$:

$\{\sigma_{22}, A^{14}\}\ P_f^{22}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{22}, \mathsf{true}\}\ P_c^{22}\ \{\sigma_{22}, \mathsf{true}\}$ and $\{\sigma_{23}, A^{14}\}\ P_f^{23}\ \{\sigma_h, \mathsf{true}\}$
or
$\{\sigma_{22}, p_{12} \square r_1\}\ P_c^{22}\ \{\sigma_{22}, \mathsf{true}\}$ and $\{\sigma_{23}, A^8\}\ P_f^{23}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{22} \triangleq At\_cr[1] \square 1 \square c_{21} \square \square 3 \square \square \mathsf{lbf}(\ldots)$. It is clear $P_c^{22} \rightarrow \mathsf{true}$. Further, as $ts[1] \square 2$, $ts[2] \square 3$ and $At\_cr[1] \square 1$, $P_c^{22} \rightarrow p_{12} \square r_1$. Then we deduce the triple $\{\sigma_{23}, A^8\}\ P_f^{23}\ \{\sigma_h, \mathsf{true}\}$.

At state s_{23}, the program is $P_f^{23} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((At\_cr[1] \square 0 \square req[1] \square 0 \square R[1,1] \square 0 \square R[1,2] \square 0 \square \varepsilon; P_1^{18}; P_1^1) \square (result[1,2] \square 3 \square c_{21} \square \square \square \varepsilon; P_1^{22}; P_1^2) \square (P_2^{15}; P_2^1) \square (P_2^{21}; P_2^2))$:

$\{\sigma_{23}, A^8\}\ P_f^{23}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{23}, p_{12}\}\ P_c^{23}\ \{\sigma_{23}, p_{12}\}$ and $\{\sigma_{24}, A^{14}\}\ P_f^{24}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{23} \triangleq At\_cr[1] \square 0 \square req[1] \square 0 \square R[1,1] \square 0 \square R[1,2] \square 0 \square c_{21} \square \square \square result[1,2] \square 3 \square \mathsf{lbf}(\ldots)$. It is obvious $ts[1] \square 2 \square ts[2] \square 3$ can indicate $P_c^{23} \rightarrow p_{12}$.

At state s_{24}, the program is $P_f^{24} \triangleq \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{11}; P_1^1) \square (hts[1,2] \square 3 \square c_{12} \square \square -1 \square \square \varepsilon; P_1^2) \square (P_2^{15}; P_2^1) \square (P_2^{21}; P_2^2))$:

$\{\sigma_{24}, A^{14}\}\ P_f^{24}\ \{\sigma_h, \mathsf{true}\}$ □□ $\{\sigma_{24}, \mathsf{true}\}\ P_c^{24}\ \{\sigma_{24}, \mathsf{true}\}$ and $\{\sigma_{25}, A^{14}\}\ P_f^{25}\ \{\sigma_h, \mathsf{true}\}$

where $P_c^{24} \triangleq hts[1,2] \square 3 \square c_{12} \square \square -1 \square \square \mathsf{lbf}(\ldots)$ and $P_f^{25} \triangleq P_f^{14}$. Clearly we have $P_c^{24} \rightarrow \mathsf{true}$, that is the present component of program can satisfy the present component of property at state s_{24}.

From above, we can observe that: (1) the values of variables at state s_{24} indeed accords with Lemma 4.1; (2) program P_f^{25} at state s_{25} is the same as P_f^{14} at state s_{14}; (3) at state s_{25}, the property is A^{14}, which implies $A^{12} \square A^{14} \square A^{16}$ (Note that the disjunctions of the formula are caused by different branches. We only demonstrate one of them and others can be obtained analogously). Therefore, the lemma holds. □

Proof of Lemma 4.2:

Proof. We prove the lemma by mathematical induction on k.

Base: This can be easily acquired by Lemma 4.1. In particular, let $k \square 1$ for (1) and (3) and let $k \square 0$ and $k \square 1$ for (2).

Induction: We assume (1), (2) and (3) respectively hold at state $s_{13+11\square k}$ ($k \geq 1$), $s_{14+11\square k}$ ($k \geq 0$) and $s_{14+11\square k}$ ($k \geq 1$). That is at state $s_{13+11\square k}$, $max[1] \square 2k-1$, $max[2] \square 2k$, $hts[1,2] \square 2k+1$, $hts[2,1] \square 2k$, $ts[1] \square 2k$, $ts[2] \square 2k+1$, $req[1] \square 0$, $req[2] \square 1$, $rd[1,2] \square 0$, $rd[2,1] \square 0$, $At\_cr[1] \square 0$, $At\_cr[2] \square 0$, $R[1,2] \square 0$, $R[1,1] \square 0$, $R[2,1] \square 0$, $R[2,2] \square 1$, $c_{12} \square \square -1 \square$, $c_{21} \square \square \square$. Then at state $s_{14+11\square k}$, the program is P_f^{14} and the property is $A^{12} \square A^{14} \square A^{16}$. We need to prove them for $k+1$. By assumption, at state $s_{14+11\square k}$, we have:

$\{\sigma_{14+11\square k}, A^{12} \square A^{14} \square A^{16}\}\ P_f^{14+11\square k} \square P_f^{14}\ \{\sigma_h, \mathsf{true}\}$

We select the triple $\{\sigma_{14+11k}, A^{12}\}\ P_f^{14}\ \{\sigma_h, \mathsf{true}\}$ to deduce.

At state s_{14+11k}:

$$\{\sigma_{14+11k}, A^{12}\}\ P_f^{14+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{14+11k}, \mathsf{true}\}\ P_c^{14+11k}\ \{\sigma_{14+11k}, \mathsf{true}\} \text{ and } \{\sigma_{15+11k}, A^{12}\}\ P_f^{15+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{14+11k} \boxminus req[1] \square 1 \square R[1,1] \square 1 \square result[2,1] \square -1 \square c_{12} \square \square\square \square \mathsf{lbf}(\ldots)$. It is obvious $P_c^{14+11k} \rightarrow \mathsf{true}$. Then the present component of program satisfies the present component of property at state s_{14+11k}.

At state s_{15+11k}, the program is $P_f^{15+11k} \boxminus \mathsf{lbf}(\ldots)\square\mathsf{frame}(\ldots)\square((max[1] \square 2k+1 \square \varepsilon\ ;P_1^{13}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (P_2^{15}\ ;P_2^1) \square (R[2,1] \square 1 \square \varepsilon\ ;P_2^2))$:

$$\{\sigma_{15+11k}, A^{12}\}\ P_f^{15+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{15+11k}, \mathsf{true}\}P_c^{15+11k}\{\sigma_{15+11k}, \mathsf{true}\} \text{ and } \{\sigma_{16+11k}, A^{12}\}P_f^{16+11k}\{\sigma_h, \mathsf{true}\}$$

where $P_c^{15+11\square k} \boxminus R[2,1] \square 1 \square max[1] \square 2k+1 \square \mathsf{lbf}(\ldots)$. Clearly $P_c^{15+11\square k} \rightarrow \mathsf{true}$.

At state s_{16+11k}, the program is $P_f^{16+11k} \boxminus \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((ts[1] \square 2k+2 \square \varepsilon\ ;P_1^{14}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (At\_cr[2] \square 1 \square \varepsilon\ ;P_2^{17}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$$\{\sigma_{16+11k}, A^{12}\}\ P_f^{16+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{16+11k}, \mathsf{true}\}\ P_c^{16+11k}\ \{\sigma_{16+11k}, \mathsf{true}\} \text{ and } \{\sigma_{17+11k}, A^{12}\}\ P_f^{17+11k}\ \{\sigma_h, \mathsf{true}\} \text{ or}$$
$$\{\sigma_{16+11k}, \neg p_{12} \square r_2\}\ P_c^{16+11k}\ \{\sigma_{16+11k}, \neg p_{12} \square r_2\} \text{ and } \{\sigma_{17+11k}, A^9\}\ P_f^{17+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{16+11k} \boxminus ts[1] \square 2k+2\square At\_cr[2] \square 1\square\mathsf{lbf}(\ldots)$. It is clear $P_c^{16+11k} \rightarrow \mathsf{true}$. Further, $ts[1] \square 2k+2\square ts[2] \square 2k+1 \square At\_cr[2] \square 1$, thus $P_c^{16+11k} \rightarrow \neg p_{12} \square r_2$. Then we choose the triple $\{\sigma_{17+11k}, A^9\}\ P_f^{17+11k}\ \{\sigma_h, \mathsf{true}\}$ to deduce.

At state s_{17+11k}, the program is $P_f^{17+11k} \boxminus \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((c_{12} \square \square 2k+2\square\square \varepsilon\ ;P_1^{15}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (At\_cr[2] \square 0 \square req[2] \square 0 \square R[2,1] \square 0 \square R[2,2] \square 0 \square \varepsilon\ ;P_2^{18}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$$\{\sigma_{17+11k}, A^9\}\ P_f^{17+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{17+11k}, \neg p_{12}\}\ P_c^{17+11k}\ \{\sigma_{17+11k}, \neg p_{12}\} \text{ and } \{\sigma_{18+11k}, A^{12}\}\ P_f^{18+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{17+11k} \boxminus c_{12} \square \square 2k+2\square\ At\_cr[2] \square 0 \square req[2] \square 0 \square R[2,1] \square 0 \square R[2,2] \square 0 \square \mathsf{lbf}(\ldots)$. Obviously $ts[1] \square 2k+2 \square ts[2] \square 2k+1$, so $P_c^{17+11k} \rightarrow \neg p_{12}$.

At state s_{18+11k}, the program is $P_f^{18+11k} \boxminus \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (P_2^{11}\ ;P_2^1) \square (result[2,1] \square 2k+2 \square c_{12} \square \square\square \varepsilon\ ;P_2^{22}\ ;P_2^2))$:

$$\{\sigma_{18+11k}, A^{12}\}\ P_f^{18+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{18+11k}, \mathsf{true}\}\ P_c^{18+11k}\ \{\sigma_{18+11k}, \mathsf{true}\} \text{ and } \{\sigma_{19+11k}, A^{12}\}\ P_f^{19+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{18+11k} \boxminus result[2,1] \square 2k+2 \square c_{12} \square \square\square \mathsf{lbf}(\ldots)$. Clearly $P_c^{18+11k} \rightarrow \mathsf{true}$.

At state s_{19+11k}, the program is $P_f^{19+11k} \boxminus \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (P_1^{21}\ ;P_1^2) \square (req[2] \square 1 \square R[2,2] \square 1 \square \varepsilon\ ;P_2^{12}\ ;P_2^1) \square (hts[2,1] \square 2k+2 \square c_{21} \square \square-1\square\square \varepsilon\ ;P_2^2))$:

$$\{\sigma_{19+11k}, A^{12}\}\ P_f^{19+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{19+11k}, \mathsf{true}\}\ P_c^{19+11k}\ \{\sigma_{19+11k}, \mathsf{true}\} \text{ and } \{\sigma_{20+11k}, A^{12}\}\ P_f^{20+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{19+11k} \boxminus req[2] \square 1\square R[2,2] \square 1\square hts[2,1] \square 2k+2\square c_{21} \square \square-1\square\square \mathsf{lbf}(\ldots)$. It is obvious $P_c^{19+11k} \rightarrow \mathsf{true}$.

At state s_{20+11k}, the program is $P_f^{20+11k} \boxminus \mathsf{lbf}(\ldots) \square \mathsf{frame}(\ldots) \square ((P_1^{15}\ ;P_1^1) \square (result[1,2] \square -1 \square c_{21} \square \square\square \varepsilon\ ;P_1^{22}\ ;P_1^2) \square (max[2] \square 2k+2 \square \varepsilon\ ;P_2^{13}\ ;P_2^1) \square (P_2^{21}\ ;P_2^2))$:

$$\{\sigma_{20+11k}, A^{12}\}\ P_f^{20+11k}\ \{\sigma_h, \mathsf{true}\}$$
$$\square\square\ \{\sigma_{20+11k}, \mathsf{true}\}\ P_c^{20+11k}\ \{\sigma_{20+11k}, \mathsf{true}\} \text{ and } \{\sigma_{21+11k}, A^{12}\}\ P_f^{21+11k}\ \{\sigma_h, \mathsf{true}\}$$

where $P_c^{20+11k} \boxminus result[1,2] \square -1 \square c_{21} \square \square\square max[2] \square 2k+2 \square \mathsf{lbf}(\ldots)$. Clearly $P_c^{20+11k} \rightarrow \mathsf{true}$.

At state s_{21+11k}, the program is $P_f^{21+11k} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((P_1^{15}\ ;P_1^1) \Box (R[1,2] \Box 1 \Box \varepsilon\ ;P_1^2) \Box (ts[2] \Box 2k+3 \Box \varepsilon\ ;P_2^{14}\ ;P_2^1) \Box (P_2^{21}\ ;P_2^2))$:

$$\begin{array}{l}\{\sigma_{21+11k}, A^{12}\}\ P_f^{21+11k}\ \{\sigma_h, \mathsf{true}\}\\ \Box\Box\ \{\sigma_{21+11k}, \mathsf{true}\}\ P_c^{21+11k}\ \{\sigma_{21+11k}, \mathsf{true}\} \text{ and } \{\sigma_{22+11k}, A^{12}\}\ P_f^{22+11k}\ \{\sigma_h, \mathsf{true}\}\end{array}$$

where $P_c^{21+11k} \triangleq R[1,2] \Box 1 \Box ts[2] \Box 2k+3 \Box \mathsf{lbf}(\ldots)$. It is clear $P_c^{21+11k} \rightarrow \mathsf{true}$.

At state s_{22+11k}, the program is $P_f^{22+11k} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((At\_cr[1] \Box 1 \Box \varepsilon\ ;P_1^{17}\ ;P_1^1) \Box (P_1^{21}\ ;P_1^2) \Box (c_{21} \Box \Box 2k+3 \Box\Box \varepsilon\ ;P_2^{15}\ ;P_2^1) \Box (P_2^{21}\ ;P_2^2))$:

$$\begin{array}{l}\{\sigma_{22+11k}, A^{12}\}\ P_f^{22+11k}\ \{\sigma_h, \mathsf{true}\}\ \Box\Box\\ \{\sigma_{22+11k}, \mathsf{true}\}\ P_c^{22+11k}\ \{\sigma_{22+11k}, \mathsf{true}\} \text{ and } \{\sigma_{23+11k}, A^{12}\}\ P_f^{23+11k}\ \{\sigma_h, \mathsf{true}\} \text{ or}\\ \{\sigma_{22+11k}, p_{12} \Box r_1\} P_c^{22+11k} \{\sigma_{22+11k}, p_{12} \Box r_1\} \text{ and } \{\sigma_{23+11k}, A^{14} \Box A^{16} \Box A^{13} \Box A^8 \Box A^7\}\ P_f^{23+11k}\ \{\sigma_h, \mathsf{true}\}\end{array}$$

where $P_c^{22+11k} \triangleq At\_cr[1] \Box 1 \Box c_{21} \Box \Box 2k+3 \Box\Box \mathsf{lbf}(\ldots)$. Obviously $P_c^{22+11k} \rightarrow \mathsf{true}$. Moreover, as $ts[1] \Box 2k+2 \Box ts[2] \Box 2k+3 \Box At\_cr[1] \Box 1$, $P_c^{22+11k} \rightarrow p_{12} \Box r_1$ holds. Then we select the triple $\{\sigma_{23+11k}, A^{13}\}\ P_f^{23+11k}\ \{\sigma_h, \mathsf{true}\}$ to deduce.

At state s_{23+11k}, the program is $P_f^{23+11k} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((At\_cr[1] \Box 0 \Box req[1] \Box 0 \Box R[1,1] \Box 0 \Box R[1,2] \Box 0 \Box \varepsilon\ ;P_1^{18}\ ;P_1^1) \Box (result[1,2] \Box 2k+3 \Box c_{21} \Box \Box\Box \varepsilon\ ;P_1^{22}\ ;P_1^2) \Box (P_2^{15}\ ;P_2^1) \Box (P_2^{21}\ ;P_2^2))$:

$$\begin{array}{l}\{\sigma_{23+11k}, A^{13}\}\ P_f^{23+11k}\ \{\sigma_h, \mathsf{true}\}\\ \Box\Box\ \{\sigma_{23+11k}, p_{12}\}\ P_c^{23+11k}\ \{\sigma_{23+11k}, p_{12}\} \text{ and } \{\sigma_{24+11k}, A^{16}\}\ P_f^{24+11k}\ \{\sigma_h, \mathsf{true}\}\end{array}$$

where $P_c^{23+11k} \triangleq At\_cr[1] \Box 0 \Box req[1] \Box 0 \Box R[1,1] \Box 0 \Box R[1,2] \Box 0 \Box c_{21} \Box \Box\Box result[1,2] \Box 2k+3 \Box \mathsf{lbf}(\ldots)$. Clearly $ts[1] \Box 2k+2 \Box ts[2] \Box 2k+3$, which leads to $P_c^{23+11k} \rightarrow p_{12}$.

At state s_{24+11k}, the program is $P_f^{24+11k} \triangleq \mathsf{lbf}(\ldots) \Box \mathsf{frame}(\ldots) \Box ((P_1^{11}\ ;P_1^1) \Box (hts[1,2] \Box 2k+3 \Box c_{12} \Box \Box -1\Box \Box \varepsilon\ ;P_1^2) \Box (P_2^{15}\ ;P_2^1) \Box (P_2^{21}\ ;P_2^2))$:

$$\begin{array}{l}\{\sigma_{24+11k}, A^{16}\}\ P_f^{24+11k}\ \{\sigma_h, \mathsf{true}\}\\ \Box\Box\ \{\sigma_{24+11k}, \mathsf{true}\}\ P_c^{24+11k}\ \{\sigma_{24+11k}, \mathsf{true}\} \text{ and } \{\sigma_{25+11k}, A^{16}\}\ P_f^{25+11k}\ \{\sigma_h, \mathsf{true}\}\end{array}$$

where $P_c^{24+11k} \triangleq hts[1,2] \Box 2k+3 \Box c_{12} \Box \Box -1\Box \Box \mathsf{lbf}(\ldots)$ and $P_f^{25+11k} \equiv P_f^{14}$. Obviously $P_c^{24+11k} \rightarrow \mathsf{true}$.

From the above deduction, we can see that: (1) at state $s_{24+11k} \Box s_{13+11\Box(k+1)}$, $max[1] \Box 2k+1 \Box 2(k+1)-1$, $max[2] \Box 2k+2 \Box 2(k+1)$, $hts[1,2] \Box 2k+3 \Box 2(k+1)+1$, $hts[2,1] \Box 2k+2 \Box 2(k+1)$, $ts[1] \Box 2k+2 \Box 2(k+1)$, $ts[2] \Box 2k+3 \Box 2(k+1)+1$, $req[1] \Box 0$, $req[2] \Box 1$, $rd[1,2] \Box 0$, $rd[2,1] \Box 0$, $At\_cr[1] \Box 0$, $At\_cr[2] \Box 0$, $R[1,1] \Box 0$, $R[1,2] \Box 0$, $R[2,1] \Box 0$, $R[2,2] \Box 1$, $c_{12} \Box \Box -1\Box$, $c_{21} \Box \Box\Box$; (2) at state $s_{25+11\Box k} \Box s_{14+11\Box(k+1)}$, the program to deduce is P_f^{14}; (3) at state $s_{25+11\Box k} \Box s_{14+11\Box(k+1)}$, the property is A^{16}, which implies the property is $A^{12} \Box A^{14} \Box A^{16}$ at state $s_{14+11\Box(k+1)}$. Hence Lemma 4.2 is correct.

References

[AFG+10] Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. Commun ACM 53(4):50–58

[BFG+90] Barringer H, Fisher M, Gabbay D, Gough G, Owens R (1990) METATEM: a framework for programming in temporal logic. In: Proceedings on Stepwise refinement of distributed systems: models, formalisms, correctness, REX workshop. Springer-Verlag New York, Inc., New York, pp 94–129

[BL84] Bledsoe W, Loveland D (1984) Automating theorem proving: after 25 years. American Mathematical Society, Providence

[Bru96] Bruns G (1996) Distributed systems analysis with CCS. Prentice Hall PTR, Englewood Cliffs

[CE81] Clarke EM, Emerson EA (1981) Design and synthesis of synchronization skeletons using branching timed temporal logic. In: LNCS, vol 131. Springer, Bertin, pp 52–71

[CGK+13] Cranen S, Groote J, Keiren JJA, Stappers FPM, Vink EP, Wesselink W, Willemse TA (2013) An overview of the mCRL2 toolset and its recent advances. In: Piterman N, Smolka SA (eds) Tools and algorithms for the construction and analysis of systems, vol 7795. Lecture notes in computer science, Springer, Berlin, pp 199–213

[CGP08] Clarke, EM, Grumberg O, Peled D (2008) Model checking. The MIT Press, Cambridge

[CY83] Chen B-S, Yeh T (1983) Formal specification and verification of distributed systems. Trans Soft Eng SE-9(6):710–722

[D13] Déharbe, D (2013) Integration of SMT-solvers in B and Event-B development environment. Sci Comput Progr 78(3):310–326

[DKH94] Duan Z, Koutny M, Holt C (1994) Projection in temporal logic programming. In: Proceedings of logic programming and automated reasoning. LNAI, vol 822, pp 333–344
[DSL13] Dong J, Sun J, Liu Y (2013) Build your own model checker in one month. In: Poceedings of ICSE13, pp 1481–1483
[DT08] Duan Z, Tian C (2008) A unified model checking approach with projection temporal logic. In: Proceedings of ICFEM08, pp 167–186
[Dua96] Duan Z (1996) An extended interval temporal logic and a framing technique for temporal logic programming. PhD thesis, University of Newcastle Upon Tyne, May 1996
[Dua06] Duan Z (2006) Temporal logic and temporal logic programming language. Science Press, Beijing
[Fis94] Fisher M (1994) A survey of concurrent metatem: the language and its applications. In: Temporal logic. Lecture notes in computer science, vol 827. Springer, Berlin, pp 480–505
[Hen07] Hennessy M (2007) A distributed Pi-calculus. Cambridge University Press, Cambridge
[Hoa78] Hoare CAR (1978) Communicating sequential processes. Commun ACM 21:666–677
[Jen91] Jensen K (1991) Coloured petri nets: a high level language for system design and analysis. In: Rozenberg G (ed) Advances in petri nets 1990, vol 483. Lecture notes in computer science, Springer, Berlin, pp 342–416
[Jon81] Jones CB (1981) Development methods for computer programs including a notion of interference. PhD thesis, Oxford University
[Lam94] Lamport L (1994) The temporal logic of actions. ACM Trans Program Lang Syst 16(3):872–923
[LT87] Lynch NA, Tuttle MR (1987) Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the sixth annual ACM symposium on principles of distributed computing. PODC '87, pp 137–151
[Mil82] Milner R (1982) A calculus of communicating systems. Springer-Verlag New York, Inc., Secaucus
[Mil99] Milner R (1999) Communicating and mobile systems: the π-calculus. Cambridge University Press, Cambridge
[Mos86] Moszkowski BC (1986) Executing temporal logic programs. PhD thesis, Cambridge Uniersity, Cambridge
[MP92] Manna Z, Pnueli A (1992) Temporal logic of reactive and concurrent systems. Springer, Berlin
[MWD11] Mo D, Wang X, Duan Z (2011) Asynchronous communication in MSVL. In: Proceeding of ICFEM2011. LNCS, vol 6991, pp 82–97
[Pet77] Peterson JL (1977) Petri nets. ACM Comput Surv 9(3):223–252
[Pnu77] Pnueli A (1977) The temporal logic of programs. In: Proceedings of the 18th annal IEEE symposium on foudations of computer science. IEEE Computer Society, pp 46–57
[RA81] Ricart G, Agrawala AK (1981) An optimal algorithm for mutual exclusion in computer networks. Commun ACM 24(1):9–17
[RNP13] Rodriguez-Navas G, Proenza J (2013) Using timed automata for modeling distributed systems with clocks: challenges and solutions. IEEE Trans Softw Eng 39(6):857–868
[Tan83] Tang CS (1983) Toward a unified logic basis for programming languages. In: Proceedings of IFIP congress. Elsevier Science, North Holland, pp 425–429
[TD11] Tian C, Duan Z (2011) Expressiveness of propositional projection temporal logic with star. Theor Comput Sci 412:1729–1744
[WLBF09] Woodcock J, Larsen PG, Bicarregui J, Fitzgerald J (2009) Formal methods: practice and experience. Comput Surv 41(4):19:1–19:36
[YDM10] Yang X, Duan Z, Ma Q (2010) Axiomatic semantics of projection temporal logic programs. Math Struct Comput Sci 20(5):865–914

Received 22 February 2014
Accepted in revised form 5 June 2014 by Dong Jin Song
Published online 29 July 2014