Formal specification and compositional verification of an atomic broadcast protocol

Eindhoven University of Technology

Department of Mathematics and Computing Science

Formal Specification and Compositional Verification of an Atomic Broadcast Protocol

by

P. Zhou and J. Hooman

94/05

Computing Science Note 94/05
Eindhoven, January 1994

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology.
Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review.
Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. M. Philips
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

Formal Specification and Compositional Verification of an Atomic Broadcast Protocol

P. Zhou J. Hooman
Dept. of Mathematics and Computing Science
Eindhoven University of Technology
P.O.Box 513
5600 MB Eindhoven, The Netherlands

January 26, 1994

Abstract

We apply formal methods to specify and verify an atomic broadcast protocol. The protocol is implemented by replicating a server process on all processors in a network. We show that the verification of the protocol can be done compositionally by using specifications in which timing is expressed by local clock values. The requirements of the protocol are formally described. Underlying communication mechanism, clock synchronization assumption, and failure assumptions are axiomatized. The server process is also represented by a formal specification. We verify that parallel execution of the server processes leads to the desired properties, by proving that the conjunction of all server specifications and axioms about the system implies the requirements of the protocol.

1 Introduction

Computing systems are composed of hardware and software components which can fail. Component failures can lead to unanticipated behaviour and service unavailability. To achieve high availability of a service despite failures, a key idea is to implement the service by a group of server processes running on distinct processors [Cri90]. Replication of service state information among group members enables the group to provide the service even when some of its members fail, since the remaining members have enough information about the service state to continue to provide it. To maintain the consistency of these replicated global states, any state update must be broadcast to all correct servers such that all these servers observe the same sequence of state updates. Thus a communication service is needed so that client processes can use it to deliver updates to their peers. This communication service is called *atomic* or *reliable* broadcast. We will refer to it as *atomic broadcast.* There are two sets of atomic broadcast protocols: *synchronous* protocols, such as [BD85,CASD85], and [Cri90], and *asynchronous* protocols, such as [BJ87] and [CM84].

Synchronous atomic broadcast protocols assume that the underlying communication delays between correct processors are bounded. Given this assumption, local clocks of correct processors can be synchronized [CAS86,CAS93]. Then the properties of synchronous atomic broadcast protocols are described in terms of local clocks as follows [CASD85,CASD89]:

- Termination: every update whose broadcast is initiated by a correct processor at time T on its clock is delivered by all correct processors at time $T + \Delta$ on their own clocks, where Δ is a positive parameter and is called *broadcast termination time.*

- Atomicity: if a correct processor delivers an update at time U on its clock, then that update was initiated by some processor and is delivered by each correct processor at time U on its own clock.
- Order: all correct processors deliver their updates in the same order.

Synchronous atomic broadcast protocols provide an upper bound for broadcast termination time. Thus they can be used in real-time applications where deadlines must always be met, even in the presence of failures. On the other hand, asynchronous broadcast protocols do not assume bounded message transmission delays between correct processors. Thus they cannot guarantee a bound for the broadcast termination time. Therefore asynchronous atomic broadcast protocols cannot be used in critical real-time applications.

In order to provide service despite the presence of faults, real-time systems often adopt fault-tolerance techniques. To achieve fault-tolerance, some kind of redundancy is introduced which will affect the timing behavior of a system. Hence it is a challenging problem to guarantee the correctness of real-time and fault-tolerant systems. We are interested in applications of formal verification methods to these systems. Since atomic broadcast service is one of the fundamental issues in fault-tolerance, we select an atomic broadcast protocol presented in [CASD85,CASD89] which tolerantes omission failures as our verification example. Henceforth, we use the term *atomic broadcast protocol* to refer to this protocol. An informal description of the protocol, an implementation, and an informal proof which shows that the implementation indeed satisfies the requirement of the protocol are presented in these papers. We follow the ideas of [CASD89] as closely as possible and compare our results with it in section 8.

The configuration of the service is illustrated in the following figure (fig.1).

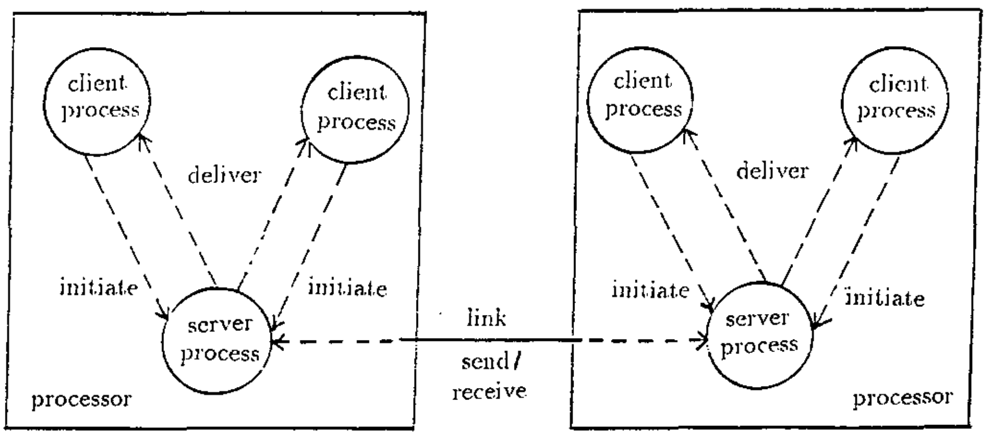

Fig.1. Atomic Broadcast Service Configuration.

The atomic broadcast service is implemented by replicating a server process on all distributed processors in a network. Thus any client process on any processor can use this service. We allow more than one client process located on one processor. Assume that there are n processors in the network. Pairs of processors are connected by links which are point-to-point, bi-directional, communication channels. The duration of message transmission between correct processors takes finite time. Each processor has access to a local clock. It is assumed that local clocks of correct processors are approximately synchronized. It is also assumed that only omission failures occur on processors and links. When a processor suffers an omission failure, it cannot send messages to other processors. When a link suffers an omission failure, the messages traveling along this link may be lost. To send an update to its peers, a client process initiates the atomic broadcast server process located on the same processor to atomically broadcast that update. After such a request, each server process will deliver that update to the client processes located on the same processor. To achieve the order property of the service, there is a priority ordering among all processors. If two updates are initiated at different clock times, they will be delivered according to the ordering of their initiation times. If they are initiated at the same clock time on different processors, they will be delivered according to the priority of their initiation processors.

In general, to formally verify a system, we need a proof theory which consists of axioms and rules about the system components. To be able to abstract from implementation details, it is often convenient to have a compositional verification method. Compositionality enables us to verify a system by using only specifications of its components without knowing any internal information of those components. Such compositional proof systems have been developed for non-real-time systems, e.g. [Zwi89], and real-time systems, such as [Hoo91] and [ZH92]. In particular, if the system is composed of parallel components, the proof method should contain a *parallel composition rule*. Let $S(p)$ denote the atomic broadcast server process running on processor p, φ denote a specification written in a formal language based on first-order logic, and $S(p)$ **sat** φ denote that server process $S(p)$ satisfies specification φ. Under the condition of maximal parallelism (i.e., each process runs at its own processor), the parallel composition rule states that if server process $S(p_i)$ satisfies specification φ_i and φ_i only refers to the interface of p_i, for $i = 1, 2, \ldots, n$, then the parallel program $S(p_1)\|\cdots\|S(p_n)$ satisfies $\bigwedge_{i=1}^{n} \varphi_i$. This rule is formalized as follows.

Parallel Composition Rule

$$\frac{S(p_i)\ \mathbf{sat}\ \varphi_i,\ \varphi_i \text{ only refers to the interface of } p_i, \text{ for } i = 1, \ldots, n}{S(p_1)\|\cdots\|S(p_n)\ \mathbf{sat}\ \bigwedge_{i=1}^{n} \varphi_i}$$

We also need a *consequence rule* to weaken a specification and a *conjunction rule* to take the conjunction of specifications. Let S be any process.

Consequence Rule $$\frac{S\ \mathbf{sat}\ \varphi,\ \varphi \rightarrow \psi}{S\ \mathbf{sat}\ \psi}$$

Conjunction Rule $$\frac{S\ \mathbf{sat}\ \varphi_1,\ S\ \mathbf{sat}\ \varphi_2}{S\ \mathbf{sat}\ \varphi_1 \wedge \varphi_2}$$

Recall that local clocks of correct processors are approximately synchronized. We show that the verification of the protocol can be done compositionally by using specifications in which timing is expressed by local clock values as follows.

- In section 2, we specify the requirements of the protocol in a formal language based on first-order logic. We call this the *top-level specification* and denote it by ABS. Thus our aim is to prove $S(p_1)\|\cdots\|S(p_n)$ **sat** ABS.

- In section 3, we axiomatize the required assumptions about the system, including underlying communication mechanism, clock synchronization assumption, and failure assumptions. We denote the conjunction of all these axioms by AX.

- In section 4, we define the properties of the atomic broadcast server process running on processor p. We call this the *server process specification* and denote it by $Spec(p)$. $Spec(p)$ should only refer to the interface of p. We assume $S(p)$ **sat** $Spec(p)$.

- By the parallel composition rule, we obtain $S(p_1)\|\cdots\|S(p_n)$ **sat** $\bigwedge_{i=1}^{n} Spec(p_i)$. By the conjunction rule, we obtain $S(p_1)\|\cdots\|S(p_n)$ **sat** $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$. We prove $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX \rightarrow ABS$ in sections 5, 6, and 7. Hence the consequence rule leads to $S(p_1)\|\cdots\|S(p_n)$ **sat** ABS.

- We compare our results with [CASD89] and conclude in section 8.

2 Top-Level Specification

We formalize the top-level requirements of the atomic broadcast protocol in this section.

Let P be a set of processor names and L a set of link names. We assume that all processors and links have unique names. We use $p, q, r, s, \ldots$ to denote elements of P and $l, l_1, \ldots$ to denote elements of L. Let G be the network of processors and links, i.e., $G = P \cup L$.

To denote real times, we use a dense time domain called $RTIME$. The standard arithmetic operators $+$, $-$, $\times$, and the relations $=$, $<$, and $\leq$ are defined on $RTIME$. We use lower case letters, e.g. t, u, v, ..., to denote variables ranging over $RTIME$.

Each processor has access to a local clock. We denote by C_p a function which represents the value of the local clock of processor p, i.e., $C_p(t)$ is the value of the local clock of p at real time t. Let all clock values range over a domain called $CVAL$. We assume $T \geq 0$, for any $T \in CVAL$. Similarly, the operators $+$, $-$, $\times$, and relations $=$, $<$, $\leq$ are defined on $CVAL$. We use capital letters, e.g. T, U, V, ..., to denote variables ranging over $CVAL$. We also use $[U, V]$, $[U, V)$, $(U, V]$, and (U, V) to express, respectively, closed, half-open, and open intervals of clock values.

The atomic broadcast service is implemented by a group of server processes replicated on all processors in the network. When a client process initiates a server process running on processor p by sending a request of broadcasting update σ, we call p the initiator of σ and say that *p initiates σ*. Similarly, when the server process delivers an update σ to client processes, we say that *p delivers σ to client processes*.

To formally describe the properties of the protocol, we define the following primitives:

- *correct*(p) **at** t: processor p is correct at real time t.
- *correct*(l) **at** t: link l is correct at real time t.
- *initiate*(p, σ) **at** t: processor p finishes with receiving a request of broadcasting update σ from a client process located on p at real time t, i.e., p initiates σ at real time t.
- *deliver*(p, σ) **at** t: processor p starts to send update σ to client processes at real time t.

Henceforth, for any primitive φ **at** t, we define the following abbreviations:

- $correct(p) \equiv \forall t : correct(p)$ **at** t
- $correct(l) \equiv \forall t : correct(l)$ **at** t
- φ $\mathbf{at_p}$ $T \equiv \exists t : \varphi$ **at** $t \wedge C_p(t) = T$
- φ $\mathbf{by_p}$ $T \equiv \exists T_0 : \varphi$ $\mathbf{at_p}$ $T_0 \wedge T_0 \leq T$
- φ $\mathbf{before_p}$ $T \equiv \exists T_0 : \varphi$ $\mathbf{at_p}$ $T_0 \wedge T_0 < T$
- φ $\mathbf{in_p}$ $I \equiv \exists T \in I : \varphi$ $\mathbf{at_p}$ T, where $I \subseteq CVAL$.

In [CASD89], assumptions about the system are simplified. For instance, it is assumed that message processing time on a correct processor is zero. In this paper, we will take all possible times spent by a correct processor into account. Then the termination and atomicity properties can only be described by using an upper bound and an interval, respectively, instead of precise time points as in [CASD89].

2.1 Termination

The property of termination is stated as follows: every update whose broadcast is initiated by a correct processor s at clock value T will be delivered at all correct processors by clock value $T + D_1$ on their own clocks, where D_1 is a positive constant and is also the broadcast termination time.

As usual, we take the convention that any free variable occurring in a formula is universally, outermostly, quantified. Thus the termination property is formally expressed as follows:

$$TERM \equiv initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge correct(s) \wedge correct(q) \rightarrow deliver(q,\sigma)\ \mathbf{by_q}\ T + D_1$$

2.2 Atomicity

The atomicity property is described as follows: if a correct processor p delivers an update at clock value U, then that update was initiated by some processor s at some local time T and is delivered by all correct processors at some local clock value between $U - D_2$ and $U + D_2$, where D_2 is a positive constant and indicates the difference of delivery times of an update by two correct processors.

This property is formalized as follows:

$$\begin{aligned} ATOM \equiv\ & deliver(p,\sigma)\ \mathbf{at_p}\ U \wedge correct(p) \wedge correct(q) \rightarrow \\ & \exists s,T : initiate(s,\sigma)\ \mathbf{at_s}\ T \wedge deliver(q,\sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2] \end{aligned}$$

Notice that the atomicity property does not follow from the termination property, because it does not assume a correct initiator.

2.3 Order

The property of order is expressed in [CASD89] as follows: all correct processors deliver their updates in the same order. We formalize it in the following way. Let U be any clock value. If $\langle\sigma_1,\ldots,\sigma_k\rangle$ is a sequence of updates delivered by processor p before local time U, then there should exist a clock value V such that $\langle\sigma_1,\ldots,\sigma_k\rangle$ has also been delivered by any other processor q before local time V. Notice that U and V can be different. Furthermore, there is no reason to exclude the possibility that more than one update is delivered at the same time by a processor. Therefore the behavior of a processor is represented by a *set* of sequences, and simultaneous updates are modelled by including all possible interleavings.

We define the following abbreviation:

- $\neg deliver(p)\ \mathbf{in_p}\ I \equiv \neg\exists\sigma : deliver(p,\sigma)\ \mathbf{in_p}\ I$.

Let $\mathbb{N}$ denote the set of all natural numbers (including 0). Let $\mathbb{N}^+ = \mathbb{N} \setminus \{0\}$. We define $List(p, U)$ to be the set of all possible sequences of updates delivered by p before local time U as follows.

Definition 2.1 For any processor p and any clock value $U \in CVAL$, define

$$\begin{aligned} List(p,U) = \{\langle\sigma_1,\sigma_2,\ldots,\sigma_k\rangle \mid\ & \text{there exist } k \in \mathbb{N}^+, U_1, U_2, \ldots, U_k \in CVAL \text{ such that} \\ & U_1 \leq U_2 \leq \ldots \leq U_k < U, \\ & deliver(p,\sigma_i)\ \mathbf{at_p}\ U_i, \text{ for all } i = 1,2,\ldots,k, \\ & \neg deliver(p)\ \mathbf{in_p}\ (U_j, U_{j+1}), \text{ for all } j = 1,2,\ldots,k-1, \text{ and} \\ & \neg deliver(p)\ \mathbf{in_p}\ [0, U_1).\} \end{aligned}$$

The order property is formalized as follows:

$$ORDER \equiv correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p,U) \subseteq List(q,V)$$

By this property, we obtain that, for any correct processors p and q, $\forall U \exists V : List(p,U) \subseteq$

$List(q, V)$ and, simultaneously, $\forall U' \exists V' : List(q, U') \subseteq List(p, V')$. Hence p and q deliver their updates in the same order.

The top-level specification of the protocol is the conjunction of these three properties. Recall that ABS denotes the top-level specification of the atomic broadcast protocol. Thus,

$ABS \equiv TERM \wedge ATOM \wedge ORDER.$

3 System Assumptions

In this section, we axiomatize the assumptions about the system. The conjunction of all the axioms is denoted by AX.

3.1 Processors and Links

We first axiomatize the topology of the network. Define the following primitives.

- $link(l, p, q)$: l is a physical communication channel between p and q.
- $Link(p) = \{l \mid \exists q : link(l, p, q)\}$: the set of links each of which connects p with another processor.

For any p, q, and l, if $l \in Link(p)$, $l \in Link(q)$, and $p \not\equiv q$, then p and q are connected by l. This is expressed by the following axiom.

Axiom 3.1 (Link) $\quad l \in Link(p) \wedge l \in Link(q) \wedge p \not\equiv q \rightarrow link(l, p, q)$

We also assume that a link connects at most two processors.

Axiom 3.2 (Point-to-Point) $\quad link(l, p, q) \wedge link(l, p, r) \rightarrow q \equiv r$

Let $FP = \{p \mid \neg correct(p)\}$ and $FL = \{l \mid \neg correct(l)\}$. Define $F = FP \cup FL$. Thus F denotes the set of processors and links which are not always correct. We assume that during any protocol execution there can be at most m processors that suffer omission failures, where $m \in \mathbb{N}$.

One important assumption about the network is that during any execution of the protocol all correct processors remain connected via correct links. Recall that G is the set of all processors and links, i.e., $G = P \cup L$. Then $G \setminus F = \{p \mid correct(p)\} \cup \{l \mid correct(l)\}$ and it denotes the set of correct processors and links. $G \setminus F$ can be considered as a graph in which processors are vertices and links are edges. We use $d(p, q)$ to denote the distance between p and q and we call $G \setminus F$ *connected* if and only if there exists a path between any two processors in $G \setminus F$. Now we can give the axiom for connectivity.

Axiom 3.3 (Connectivity) $\quad G \setminus F$ is connected.

Given axiom 3.3, we assume that the diameter of $G \setminus F$ is d.

3.2 Bounded Communication

Now we give the axioms for the underlying communication mechanism. We define two primitives:

- $send(p, m, l)$ **at** t: processor p starts to send message m along link l at real time t.
- $receive(p, m, l)$ **at** t: processor p finishes with receiving message m along link l at real time t.

The abbreviations defined in section 2 are also used for these two primitives.

Two processors connected by a link are called neighbors. When $send(p, m, l)$ **at** t or $receive(p, m, l)$ **at** t holds, l must be a link connecting p and one of its neighbors.

Axiom 3.4 (Neighbor) $send(p, m, l)\ \mathbf{at_q}\ T \ \vee\ receive(p, m, l)\ \mathbf{at_q}\ T \rightarrow l \in Link(p)$

Two processors can send messages to each other if they are connected by a link. Communication along links is synchronous in the sense that the duration of the transmission of a message is bounded by two parameters γ and δ with $\gamma, \delta \in CVAL$, $\gamma > 0$, and $\gamma \leq \delta$. Let p and q be two correct processors connected by a correct link l. Let r be any correct processor to be used as reference. If p sends message m along link l at clock value U according to the clock of r, then q will receive m along l at some clock value in the interval $[U+\gamma, U+\delta]$ according to the clock of r.

Axiom 3.5 (Bounded Communication)

$$send(p, m, l)\ \mathbf{at_r}\ U \wedge correct(p) \wedge correct(q) \wedge link(l, p, q) \wedge correct(l) \wedge correct(r) \rightarrow receive(q, m, l)\ \mathbf{in_r}\ [U+\gamma, U+\delta]$$

3.3 Clock Synchronization

We assume that clocks of correct processors are synchronized within a parameter ϵ.

Axiom 3.6 (Clock Synchronization)

$$correct(p)\ \mathbf{at}\ t \wedge correct(q)\ \mathbf{at}\ t \rightarrow |C_p(t) - C_q(t)| < \epsilon$$

It is trival to derive the following lemma.

Lemma 3.1 (Clock Synchronization) $correct(p) \wedge correct(q) \rightarrow |C_p(t) - C_q(t)| < \epsilon$

We also assume that local clocks are monotonic.

Axiom 3.7 (Monotonic Clock) $t_1 \leq t_2 \leftrightarrow C_p(t_1) \leq C_p(t_2)$

According to [Cri93], an implicit assumption was made and used in [CASD89], namely that any clock on a correct processor has a speed that varies from the speed of any other clock on a correct processor by a very small quantity ρ, $\rho \geq 0$. This ρ drift was neglected in [CASD89] and it resulted in the following approximation: while a message travels between two processors the clocks of the two processors will keep their distance constant. We take this ρ factor into account and formalize this assumption as follows:

Axiom 3.8 (Relative Speed) $correct(p) \wedge correct(q) \wedge t_1 \leq t_2 \rightarrow$

$$(1-\rho)(C_p(t_2) - C_p(t_1)) \leq C_q(t_2) - C_q(t_1) \leq (1+\rho)(C_p(t_2) - C_p(t_1))$$

3.4 Failure Assumptions

The atomic broadcast protocol verified in this paper tolerates omission failures. When a processor suffers an omission failure, it cannot send out messages. More precisely, if a processor p is not correct at real time t, then p is not able to send any message m along any link l at time t. This is also called the *fail silence* property of processors.

Axiom 3.9 (Fail Silence) $\neg correct(p)\ \mathbf{at_q}\ T \rightarrow \neg send(p, m, l)\ \mathbf{at_q}\ T$

When a link suffers an omission failure, the messages entrusted on that link may be lost. But if a message has been received by a processor along a (possibly faulty) link, then that message should have been correctly transmitted by that link, i.e., that message is not corrupted, there are no timing errors on the message sending and receiving, etc.. Therefore, if a processor q receives a message m along link l at clock value V, then there exists another processor p which has sent that message earlier along l at some time between $[V-\delta, V-\gamma]$ according to the clock of r.

Axiom 3.10 (Only Omission Failure)

$$receive(q,m,l)\ \mathbf{at_r}\ V \wedge correct(r) \rightarrow \exists p \not\equiv q : send(p,m,l)\ \mathbf{in_r}\ [V-\delta, V-\gamma]$$

4 Server Process Specification

In this section, we characterize $S(p)$, i.e., the atomic broadcast server process running on p.

Notice that, in the top-level specification, only delivery of updates is important and thus primitive $deliver(p,\sigma)$ **at** t is used. In the server process specification, information about the initiation time T and the initiator s of an update σ is needed to implement the top-level specification. Therefore we define another primitive $convey(p, <T,s,\sigma>)$ **at** t as follows:

- $convey(p, <T,s,\sigma>)$ **at** t: processor p starts to send message $<T,s,\sigma>$ to client processes at real time t.

Then the relation between $deliver(p,\sigma)$ **at** t and $convey(p, <T,s,\sigma>)$ **at** t is clear:

- $deliver(p,\sigma)\ \mathbf{at}\ t \leftrightarrow \exists s,T : convey(p, <T,s,\sigma>)\ \mathbf{at}\ t$

Assume that any correct processor can send a message to all its neighbors within $T_s \in CVAL$ time units and any correct processor can convey all the updates initiated at the same clock time to client processes within $T_c \in CVAL$ time units. Let $T_r \in CVAL$, $T_r \geq T_s$, be the time to ensure that all correct processors have received a message containing an update after it is initiated. These parameters will be used to determine the values of D_1 and D_2 occurring in the top-level specification.

The server specification is described as follows.

- Initiation requirement.
  When p initiates an update σ at clock time T, it will send message $<T,p,\sigma>$ to all its neighbors immediately. When p has waited long enough to be sure that all correct processors have received that message, p will convey $<T,p,\sigma>$ to client processes. This is formalized by the following formula:

  $$Start(p) \equiv initiate(p,\sigma)\ \mathbf{at_p}\ T \rightarrow \forall l \in Link(p) : send(p, <T,p,\sigma>, l)\ \mathbf{in_p}\ [T, T+T_s] \wedge convey(p, <T,p,\sigma>)\ \mathbf{in_p}\ [T+T_r, T+T_r+T_c]$$

- Relay requirement.
  When p receives a message $<T,s,\sigma>$, it will relay this message on all links except the one along which it received this message. As in the initiator's case, when its clock reaches $T+T_r$, p will convey $<T,s,\sigma>$ to client processes.

  $$Relay(p) \equiv receive(p, <T,s,\sigma>, l)\ \mathbf{at_p}\ U \rightarrow \forall l_1 \in Link(p) \setminus \{l\} : send(p, <T,s,\sigma>, l_1)\ \mathbf{in_p}\ [U, U+T_s] \wedge convey(p, <T,s,\sigma>)\ \mathbf{in_p}\ [T+T_r, T+T_r+T_c]$$

- Convey requirement.
  If processor p conveys a message $<T,s,\sigma>$ at clock time U, then there can be only two

possibilities: either p initiated σ itself at local clock time T with $U \in [T+T_r, T+T_r+T_c]$, or p received the message $< T, s, \sigma >$ at some clock value V and $p \not\equiv s \wedge U \in [T+T_r, T+T_r+T_c]$ holds.

When p initiates σ at local time T or it receives $< T, s, \sigma >$ at some local time V, we say that p *learns of* message $< T, s, \sigma >$ and define:

$$\begin{aligned} Learn(p, < T, s, \sigma >) \equiv\ & (initiate(p, \sigma) \textbf{ at}_\mathbf{p}\ T \wedge p \equiv s) \vee \\ & (\exists l, V : receive(p, < T, s, \sigma >, l) \textbf{ at}_\mathbf{p}\ V \wedge p \not\equiv s) \end{aligned}$$

Then the requirement is formalized by the formula $Origin(p)$:

$$\begin{aligned} Origin(p) \equiv\ & convey(p, < T, s, \sigma >) \textbf{ at}_\mathbf{p}\ U \rightarrow \\ & Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c] \end{aligned}$$

- Ordering requirement.
  If two messages are conveyed by processor p, then they will be conveyed in the order of initiation times of updates contained in these two messages. If initiation times are the same, then they will be conveyed according to the priority of initiators. Therefore it is assumed that there is a total order $\prec$ on the set of processor names P. This total order specifies a priority ordering among processors. We define a lexicographical ordering $\sqsubset$ on pairs $< T, s >$.

Definition 4.1 For any two pairs (T_1, s_1) and (T_2, s_2), $(T_1, s_1) \sqsubset (T_2, s_2)$ iff $(T_1 < T_2) \vee (T_1 = T_2 \wedge s_1 \prec s_2)$.

Then the fourth requirement is formalized by the following formula $Sequen(p)$:

$$\begin{aligned} Sequen(p) \equiv\ & convey(p, < T_1, s_1, \sigma_1 >) \textbf{ at}_\mathbf{p}\ V_1 \wedge convey(p, < T_2, s_2, \sigma_2 >) \textbf{ at}_\mathbf{p}\ V_2 \\ & \rightarrow (V_1 < V_2 \leftrightarrow (T_1, s_1) \sqsubset (T_2, s_2)) \end{aligned}$$

The requirements mentioned above are only for correct processors. Since omission failures are allowed, we still need to define what is the acceptable behaviour for faulty processors. Thus we have the following requirement for any arbitrary processor p.

- Failure requirement.
  When p sends a message $< T, s, \sigma >$ to a neighbor at local time U, there can be only two possibilities: either p initiated σ itself at local time T and $U \in [T, T + T_s]$ holds, or p received $< T, s, \sigma >$ at some local time V and $U \in [V, V + T_s]$ holds.

$$\begin{aligned} Source(p) \equiv\ & send(p, < T, s, \sigma >, l) \textbf{ at}_\mathbf{p}\ U \rightarrow \\ & (initiate(p, \sigma) \textbf{ at}_\mathbf{p}\ T \wedge U \in [T, T + T_s] \wedge p \equiv s) \vee \\ & \exists l_1, V : (receive(p, < T, s, \sigma >, l_1) \textbf{ at}_\mathbf{p}\ V \wedge U \in [V, V + T_s] \wedge p \not\equiv s) \end{aligned}$$

When $send(p, < T, s, \sigma >, l) \textbf{ at}_\mathbf{p}\ U$ holds, by the fail silence axiom 3.9, $correct(p) \textbf{ at}_\mathbf{p}\ U$ holds. But $correct(p) \textbf{ at}_\mathbf{p}\ U$ does not imply $correct(p)$. It is quite possible that p is faulty at some other time. That is why this requirement should be for any processor p and not only for correct one.

Now we assume that server process $S(p)$ satisfies specification $Spec(p)$ with

$Spec(p) \equiv [correct(p) \rightarrow Start(p) \wedge Relay(p) \wedge Origin(p) \wedge Sequen(p)] \wedge Source(p)$.

Axiom 4.1 (Server Process Specification) $S(p)$ **sat** $Spec(p)$

Thus the behavior of any processor p is specified by this axiom and the fail silence axiom 3.9.

5 Verification of Termination

As explained in the Introduction, our aim is to prove $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX \rightarrow ABS$, where AX is the conjunction of all the axioms and ABS is the top-level specification of the protocol. Thus we assume $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$ and prove ABS.

In this section, we prove the termination property of the protocol. To make the proof easier, we first give some additional lemmas.

Since we have assumed $\bigwedge_{i=1}^{n} Spec(p_i) \wedge AX$, we can rewrite a part of the $Spec(p)$ to a more general form in which the clock values are measured on an arbitrary correct processor r.

Lemma 5.1 (Modified Server Specification)

$$correct(r) \rightarrow [correct(p) \rightarrow Forward(p,r)] \wedge NSource(p,r),$$

where $Forward(p,r)$ is generalized from $Relay(p)$ and formalized as

$$Forward(p,r) \equiv receive(p, <T,s,\sigma>, l) \ \mathbf{at_r} \ U \rightarrow$$
$$\forall l_1 \in Link(p) \setminus \{l\} : send(p, <T,s,\sigma>, l_1) \ \mathbf{in_r} \ [U, U+(1+\rho)T_s]$$

and $NSource(p,r)$ is a general form of $Source(p)$:

$$NSource(p,r) \equiv send(p, <T,s,\sigma>, l) \ \mathbf{at_r} \ U \rightarrow$$
$$(initiate(p,\sigma) \ \mathbf{at_p} \ T \wedge U \in (T-\epsilon, T+T_s+\epsilon) \wedge p \equiv s) \vee$$
$$\exists l_1, V : (receive(p, <T,s,\sigma>, l_1) \ \mathbf{at_r} \ V \wedge U \in [V, V+(1+\rho)T_s] \wedge p \not\equiv s)$$

Proof: We prove this lemma by two steps.

- First, we prove $correct(r) \wedge correct(p) \rightarrow Forward(p,r)$. Assume that $correct(r) \wedge correct(p) \wedge receive(p, <T,s,\sigma>, l) \ \mathbf{at_r} \ U$ holds. Let t_1 be the real time such that $C_r(t_1) = U$. Suppose $C_p(t_1) = U_1$. Then we have $receive(p, <T,s,\sigma>, l) \ \mathbf{at_p} \ U_1$. By $Relay(p)$, we obtain $\forall l_1 \in Link(p) \setminus \{l\} : send(p, <T,s,\sigma>, l_1) \ \mathbf{in_p} \ [U_1, U_1+T_s]$.
  Let t_2 be the real time such that $C_p(t_2) = U_1 + T_s$. Thus we have
  $\forall l_1 \in Link(p) \setminus \{l\} : send(p, <T,s,\sigma>, l_1) \ \mathbf{in} \ [t_1, t_2]$.
  Since $T_s \geq 0$, we obtain $C_p(t_1) \leq C_p(t_2)$. By the monotonic clock axiom 3.7, we have $t_1 \leq t_2$. Then by the relative speed axiom 3.8, we obtain
  $(1-\rho)(C_p(t_2) - C_p(t_1) \leq C_r(t_2) - C_r(t_1) \leq (1+\rho)(C_p(t_2) - C_p(t_1))$.
  Hence $C_r(t_2) \leq U + (1+\rho)T_s$. Thus we obtain
  $\forall l_1 \in Link(p) \setminus \{l\} : send(p, <T,s,\sigma>, l_1) \ \mathbf{in_r} \ [U, U+(1+\rho)T_s]$
  and then $Forward(p,r)$ holds.

- Second, we prove $correct(r) \rightarrow NSource(p,r)$. Assume $correct(r)$ and $send(p, <T,s,\sigma>, l) \ \mathbf{at_r} \ U$ hold. Let t_1 be the real time such that $C_r(t_1) = U$. Suppose $C_p(t_1) = U_1$. Then we have $send(p, <T,s,\sigma>, l) \ \mathbf{at_p} \ U_1$. By $Source(p)$, we obtain
  $$(initiate(p,\sigma) \ \mathbf{at_p} \ T \wedge U_1 \in [T, T+T_s] \wedge p \equiv s) \vee \quad (1)$$
  $$\exists l_1, V_1 : (receive(p, <T,s,\sigma>, l_1) \ \mathbf{at_p} \ V_1 \wedge U_1 \in [V_1, V_1+T_s] \wedge p \not\equiv s) \quad (2)$$
  Assume (1) holds. From $send(p, <T,s,\sigma>, l) \ \mathbf{at} \ t_1$, by the fail silence axiom 3.9, we have $correct(p) \ \mathbf{at} \ t_1$. From $correct(r)$ and the clock synchronization axiom 3.6, we obtain $|C_r(t_1) - C_p(t_1)| < \epsilon$. Since $C_p(t_1) = U_1 \in [T, T+T_s]$, we have $C_r(t_1) \in (T-\epsilon, T+T_s+\epsilon)$. From (1), we obtain
  $$initiate(p,\sigma) \ \mathbf{at_p} \ T \wedge U \in (T-\epsilon, T+T_s+\epsilon) \wedge p \equiv s \quad (3)$$
  Suppose that (2) holds. Let t_2 be the real time such that $C_p(t_2) = V_1$. Then there exists a V such that $C_r(t_2) = V$. Since $C_p(t_2) \leq C_p(t_1)$, by the monotonic clock axiom 3.7, we have $t_2 \leq t_1$. By the relative speed axiom 3.8, we have
  $(1-\rho)(C_p(t_1) - C_p(t_2) \leq C_r(t_1) - C_r(t_2) \leq (1+\rho)(C_p(t_1) - C_p(t_2))$.
  From $U_1 \in [V_1, V_1+T_s]$, we have $0 \leq C_p(t_1) - C_p(t_2) \leq T_s$ and then
  $C_r(t_2) \leq C_r(t_1) \leq C_r(t_2) + (1+\rho)T_s$, i.e., $U \in [V, V+(1+\rho)T_s]$.

From (2), we obtain

$$\exists l_1, V : (receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V \wedge U \in [V, V + (1+\rho)T_s] \wedge p \not\equiv s) \qquad (4)$$

Combining (3) and (4), we have proved $NSource(p, r)$. □

The second lemma expresses that if a correct processor p receives a message $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, then its correct neighbor q which is not s will receive $< T, s, \sigma >$ by $V + (1+\rho)T_s + \delta$ measured on the clock of r.

Lemma 5.2 (Propagation)
$receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V \wedge correct(p) \wedge correct(q) \wedge link(l_2, p, q) \wedge correct(l_2) \wedge q \not\equiv s$
$\wedge \ correct(r) \rightarrow \exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + (1+\rho)T_s + \delta$

Proof: Assume that the premise of the lemma holds. Since $receive(p, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V$ holds, there are two possibilities.

- If $l_1 \not\equiv l_2$, then q is not the processor which just sent the message $< T, s, \sigma >$ to p. By $Forward(p, r)$, p will send the message $< T, s, \sigma >$ to q along link l_2 within $(1+\rho)T_s$ time units as measured on the clock of r. Thus we have
$send(p, < T, s, \sigma >, l_2) \ \mathbf{in_r} \ [V, V + (1+\rho)T_s]$.
Then there exists an V_1 such that
$send(p, < T, s, \sigma >, l_2) \ \mathbf{at_r} \ V_1 \wedge V_1 \in [V, V + (1+\rho)T_s]$.
By the bounded communication axiom 3.5, we obtain
$receive(q, < T, p, \sigma >, l_2) \ \mathbf{in_r} \ [V_1 + \gamma, V_1 + \delta]$.
Together with $V_1 \leq V + (1+\rho)T_s$, we obtain
$\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + (1+\rho)T_s + \delta$.

- If $l_1 \equiv l_2$, then p receives $< T, p, \sigma >$ from link l_2 and thus we have
$receive(p, < T, s, \sigma >, l_2) \ \mathbf{at_r} \ V$.
By the only omission failure axiom 3.10, there exists a p_1 such that
$p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l_2) \ \mathbf{in_r} \ [V - \delta, V - \gamma]$
holds. By the neighbor axiom 3.4, we have $l_2 \in Link(p) \wedge l_2 \in Link(p_1)$. Since $p \not\equiv p_1$, by the link axiom 3.1, we obtain $link(l_2, p, p_1)$. But it is assumed that $link(l_2, p, q)$. Thus by the point-to-point axiom 3.2, we obtain $p_1 \equiv q$. Thus there exists a U such that
$send(q, < T, s, \sigma >, l_2) \ \mathbf{at_r} \ U \wedge U \in [V - \delta, V - \gamma]$
holds. Since $q \not\equiv s$, by $NSource(q, r)$, we obtain
$\exists l, V' : (receive(q, < T, s, \sigma >, l) \ \mathbf{at_r} \ V' \wedge U \in [V', V' + (1+\rho)T_s])$.
From $V' \leq U$ and $U \leq V - \gamma$, we obtain $V' \leq V - \gamma$ and thus $V' \leq V + (1+\rho)T_s + \delta$.
Thus we have $\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + (1+\rho)T_s + \delta$. □

The next lemma shows that if correct processor s initiates an update σ at local time T, then any another correct processor q will receive $< T, s, \sigma >$ by $T + d(s,q)((1+\rho)T_s + \delta)$ measured on the clock of s, where $d(s,q)$ denotes the distance between s and q.

Lemma 5.3 (Bounded Receiving)

$$initiate(s, \sigma) \ \mathbf{at_s} \ T \wedge correct(s) \wedge correct(q) \wedge q \not\equiv s \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_s} \ T + d(s,q)((1+\rho)T_s + \delta)$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between s and q. Since $s \not\equiv q$, we start with $d(s,q) = 1$.

- $d(s,q) = 1$. Since both s and q are correct processors, by the definition of $d(s,q)$, they are connected by some correct link. Let l be that link. Then we obtain $link(l, s, q) \wedge correct(l)$. Since $correct(s)$ holds, we have $Start(s)$. From $Start(s)$ and $initiate(s, \sigma) \ \mathbf{at_s} \ T$, s will send the message $< T, s, \sigma >$ to processor q along link l. Thus we have

$send(s, < T, s, \sigma >, l)$ $\mathbf{in_s}$ $[T, T + T_s]$.
By definition, there exists a U such that
$send(s, < T, s, \sigma >, l)$ $\mathbf{at_s}$ $U \wedge U \in [T, T + T_s]$.
By the bounded communication axiom 3.5, we obtain
$receive(q, < T, s, \sigma >, l)$ $\mathbf{in_s}$ $[U + \gamma, U + \delta]$.
From $U \leq T + T_s$, we obtain
$receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + T_s + \delta$.
Since $\rho \geq 0$, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s, q)((1 + \rho)T_s + \delta)$.

- $d(s, q) = k+1$ with $k \geq 1$. By definition, there must exist a link l_2 and a processor q_1 such that $link(l_2, q_1, q) \wedge correct(l_2) \wedge correct(q_1) \wedge d(s, q_1) = k \wedge d(q_1, q) = 1$ holds. By the induction hypothesis, we have $\exists l_1 : receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{by_s}$ $T + k((1 + \rho)T_s + \delta)$.
By definition, there exists a V_1 such that
$\exists l_1 : (receive(q_1, < T, s, \sigma >, l_1)$ $\mathbf{at_s}$ $V_1 \wedge V_1 \leq T + k((1 + \rho)T_s + \delta)$).
By the propagation lemma 5.2, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $V_1 + (1 + \rho)T_s + \delta$, i.e.,
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + (k + 1)((1 + \rho)T_s + \delta)$.
Hence we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s, q)((1 + \rho)T_s + \delta)$. □

The next lemma shows that if a correct processor s initiates σ at local time T, then every correct processor q will convey $< T, s, \sigma >$ in the interval $[T + T_r, T + T_r + T_c]$ according to its own clock.

Lemma 5.4 (Convey)

$$initiate(s, \sigma) \ \mathbf{at_s} \ T \wedge correct(s) \wedge correct(q) \rightarrow$$
$$convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c]$$

Proof: Assume that the premise of the lemma holds. We prove this lemma in two cases.

- $d(s, q) = 0$. By definition, we have $s \equiv q$. By $correct(s)$, we have $Start(s)$. From $initiate(s, \sigma)$ $\mathbf{at_s}$ T, we obtain
$convey(s, < T, s, \sigma >)$ $\mathbf{in_s}$ $[T + T_r, T + T_r + T_c]$. Thus we have
$convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.

- $d(s, q) > 0$. By definition, we have $s \not\equiv q$. By the bounded receiving lemma 5.3, we obtain $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_s}$ $T + d(s, q)((1 + \rho)T_s + \delta)$.
By the clock synchronization lemma 3.1, we have
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{before_q}$ $T + d(s, q)((1 + \rho)T_s + \delta) + \epsilon$.
Thus there exists a V such that $\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{at_q}$ V.
By $Relay(q)$, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$. □

Next we prove that the termination property follows from the axioms and lemmas given before.

Theorem 5.1 (Termination) If $D_1 \geq T_r + T_c$, then

$$initiate(s, \sigma) \ \mathbf{at_s} \ T \wedge correct(s) \wedge correct(q) \rightarrow deliver(q, \sigma) \ \mathbf{by_q} \ T + D_1,$$

i.e., the termination property TERM holds.

Proof: Assume that the premise of this theorem holds. By the convey lemma 5.4, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$. As observed in section 4, we have $deliver(q, \sigma)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.
Since $D_1 \geq T_r + T_c$, we have $deliver(q, \sigma)$ $\mathbf{by_q}$ $T + D_1$. □

6 Verification of Atomicity

In this section, we prove the atomicity property of the atomic broadcast protocol. We first show some lemmas which will help prove the atomicity property.

The next lemma states that if correct processor p receives message $< T, s, \sigma >$ at local time V, then that update σ was initiated by processor s at local time T.

Lemma 6.1 (Initiation)

$$receive(p, < T, s, \sigma >, l) \ \mathbf{at_p} \ V \wedge correct(p) \rightarrow initiate(s, \sigma) \ \mathbf{at_s} \ T$$

Proof: Assume that the premise of the lemma holds. By the only omission failure axiom 3.10, there exist s_1 and U_1 such that

$$s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l) \ \mathbf{at_p} \ U_1 \wedge U_1 \in [V - \delta, V - \gamma]. \quad (1)$$

By $NSource(s_1, p)$, there exist l_1 and V_1 such that

$$(initiate(s_1, \sigma) \ \mathbf{at_{s_1}} \ T \wedge s_1 \equiv s) \vee \quad (2)$$

$$(receive(s_1, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V_1 \wedge s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + (1 + \rho)T_s]). \quad (3)$$

If (2) holds, we have proved $initiate(s, \sigma) \ \mathbf{at_s} \ T$.

If (2) does not hold, then s_1 is not the initiator of σ and (3) holds.

From (1), we have $U_1 \leq V - \gamma$, i.e., $V \geq U_1 + \gamma$. From (3), we have $U_1 \geq V_1$. Thus we obtain $V \geq V_1 + \gamma$, i.e., $V - V_1 \geq \gamma$.

From $receive(s_1, < T, s, \sigma >, l_1) \ \mathbf{at_p} \ V_1$, we follow the above steps and then obtain another processor $s_2 \not\equiv s_1$. Let $k \in I\!N$, $k \geq 2$, such that $k > V/\gamma$ (notice that $\gamma > 0$). Then there are two possibilities:

- either there exists a $i < k$ such that s_i is the initiator of σ and $s_i \equiv s$. Hence we have obtained $initiate(s, \sigma) \ \mathbf{at_s} \ T$;

- or there does not exist a $i < k$ such that s_i is the initiator of σ. Thus s_1, ..., s_{k-1} are not the initiator of σ. Then, for any $i = 2, 3, \ldots, k - 1$, there exist l_i and V_i such that

  $$s_i \not\equiv s_{i-1} \ \wedge \ receive(s_i, < T, s, \sigma >, l_i) \ \mathbf{at_p} \ V_i \ \wedge \ s_i \not\equiv s \ \wedge \ V_{i-1} - V_i \geq \gamma$$

  holds. From $V_{i-1} - V_i \geq \gamma$ and $V - V_1 \geq \gamma$, we obtain $V - V_i \geq i\gamma$, for any $i = 1, 2, \ldots, k-1$. From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1}) \ \mathbf{at_p} \ V_{k-1}$, by the only omission failure axiom 3.10, there exists a processor $s_k \not\equiv s_{k-1}$ such that

  $send(s_k, < T, s, \sigma >, l_{k-1}) \ \mathbf{in_p} \ [V_{k-1} - \delta, V_{k-1} - \gamma]$ holds.

  By $NSource(s_k, p)$, there exist l_k and V_k such that

  $$(initiate(s_k, \sigma) \ \mathbf{at_{s_k}} \ T \wedge s_k \equiv s) \vee \quad (5)$$

  $$(receive(s_k, < T, s, \sigma >, l_k) \ \mathbf{at_{s_k}} \ V_k \wedge s_k \not\equiv s) \quad (6)$$

  holds. If (6) holds, similar to before, we can derive $V_{k-1} - V_k \geq \gamma$. From $V - V_i \geq i\gamma$, we obtain $V - V_k \geq k\gamma$. Since $k > V/\gamma$, we have $V - V_k > V$ and thus $V_k < 0$. Recall that all local clock values are nonnegative. Hence (6) does not hold. Therefore (5) must hold, i.e., s_k is the initiator of σ and $s_k \equiv s$. □

We define an abbreviation $Firstrec(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ V$, which expresses that p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r and p is one of the first correct processors which have received $< T, s, \sigma >$ according to the clock of r, as follows:

$$Firstrec(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ V \equiv receive(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ V \wedge correct(r) \wedge correct(p) \wedge$$
$$\forall p', l', V' : (correct(p') \wedge p' \not\equiv p \wedge receive(p', < T, s, \sigma >, l') \ \mathbf{at_r} \ V' \rightarrow V' \geq V)$$

The next lemma shows that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then any processor q which is not p and has sent $< T, s, \sigma >$ earlier than V is a faulty processor.

Lemma 6.2 (Faulty Sender)

$$Firstrec(p, < T, s, \sigma >, l_1)\ \mathbf{at_r}\ V \wedge send(q, < T, s, \sigma >, l_2)\ \mathbf{at_r}\ U \wedge p \not\equiv q \wedge \neg correct(s) \wedge U < V \rightarrow \neg correct(q)$$

Proof: Assume that the premise of the lemma holds. From $send(q, < T, s, \sigma >, l_2)\ \mathbf{at_r}\ U$, by $NSource(q, r)$, we obtain

$$(initiate(q, \sigma)\ \mathbf{at_q}\ T \wedge q \equiv s) \vee \quad (1)$$

$$\exists l', V' : (receive(q, < T, s, \sigma >, l')\ \mathbf{at_r}\ V' \wedge q \not\equiv s \wedge U \in [V', V' + (1+\rho)T_s]\ ). \quad (2)$$

Then there exist two possibilities:

- if (1) holds, then $q \equiv s$ and thus, by assumption, $\neg correct(q)$ holds;
- if (2) holds, we have $V' \leq U$. Since $U < V$, we obtain $V' < V$. If $correct(q)$ holds, by $Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at_r}\ V$, we would have $V' \geq V$ and thus it leads to a contradiction. Thus $\neg correct(q)$ holds. □

The following lemma shows that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r, p is one of the first correct processors which have received $< T, s, \sigma >$, and s is faulty, then $V < T + m((1+\rho)T_s + \delta) + \epsilon$, where m is the maximum number of faulty processors in the network.

Lemma 6.3 (First Correct Receiving)

$$Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at_r}\ V \wedge \neg correct(s) \rightarrow V < T + m((1+\rho)T_s + \delta) + \epsilon$$

Proof: Assume that the premise of the lemma holds. From $Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at_r}\ V$, we obtain $receive(p, < T, s, \sigma >, l)\ \mathbf{at_r}\ V$. By the only omission failure axiom 3.10, there exist s_1 and U_1 such that $s_1 \not\equiv p \wedge send(s_1, < T, s, \sigma >, l)\ \mathbf{at_r}\ U_1 \wedge U_1 \in [V - \delta, V - \gamma]$ holds. Thus we have

$$V \leq U_1 + \delta \text{ and } U_1 \leq V - \gamma. \quad (1)$$

Then we obtain $V \geq U_1 + \gamma$. Since $\gamma > 0$, we have

$$V > U_1. \quad (2)$$

Since $Firstrec(p, < T, s, \sigma >, l)\ \mathbf{at_r}\ V$ holds, by the faulty sender lemma 6.2, s_1 is a faulty processor, i.e., $\neg correct(s_1)$ holds. By $NSource(s_1, r)$, there exist l_1 and V_1 such that

$$(initiate(s_1, \sigma)\ \mathbf{at_{s_1}}\ T \wedge s_1 \equiv s \wedge U_1 \in (T - \epsilon, T + T_s + \epsilon)\ ) \vee \quad (3)$$

$$(receive(s_1, < T, s, \sigma >, l_1)\ \mathbf{at_r}\ V_1 \wedge s_1 \not\equiv s \wedge U_1 \in [V_1, V_1 + (1+\rho)T_s]\ ) \quad (4)$$

holds. Then there are two possibilities.

- If (3) holds, then s_1 is the initiator of σ and we have $U_1 < T + T_s + \epsilon$. Together with (1), we obtain $V < T + (1+\rho)T_s + \delta + \epsilon$. Since $\neg correct(s)$ holds, there is at least one faulty processor, i.e., the maximum number of faulty processors $m \geq 1$. Thus we obtain $V < T + m((1+\rho)T_s + \delta) + \epsilon$.

- If (4) holds, we have $U_1 \leq V_1 + (1+\rho)T_s$. From (1), we obtain

  $$V \leq V_1 + (1+\rho)T_s + \delta. \quad (5)$$

  From $receive(s_1, < T, s, \sigma >, l_1)\ \mathbf{at_r}\ V_1$, by the only omission failure axiom 3.10, there exist s_2 and U_2 such that s_2 has sent $< T, s, \sigma >$ to s_1 along link l_1 at time U_2 measured on the clock of r. Similar to before, we have $U_2 \in [V_1 - \delta, V_1 - \gamma]$, i.e., $U_2 \leq V_1 - \gamma$. From (4), $V_1 \leq U_1$ and thus $U_2 \leq U_1 - \gamma$. From (2), $U_1 < V$ and then $U_2 < V - \gamma$. Hence $V > U_2$. Then by the faulty sender lemma 6.2, $\neg correct(s_2)$ holds.
  By $NSource(s_2, r)$, we obtain a formula similar to (3) and (4).
  If s_2 is not the initiator of σ, we follow the above steps and then obtain another s_3 which is also a faulty processor. Since there are at most m faulty processors, we cannot continue this procedure infinitely. We must obtain a s_k which is the initiator of σ with $k \leq m$.

For any $i = 2, 3, \ldots, k-1$, by the only omission failure axiom 3.10 and $NSource(s_i, r)$, there exist l_i and V_i such that

$$s_i \not\equiv s_{i-1} \wedge receive(s_i, < T, s, \sigma >, l_i) \ \mathbf{at_r} \ V_i \ \wedge \ s_i \not\equiv s \wedge V_{i-1} \leq V_i + (1+\rho)T_s + \delta$$

holds. Then we obtain

$$V_1 \leq V_{k-1} + (k-2)((1+\rho)T_s + \delta). \tag{6}$$

From $receive(s_{k-1}, < T, s, \sigma >, l_{k-1}) \ \mathbf{at_r} \ V_{k-1}$, by the only omission failure axiom 3.10, there exists a U_k such that

$s_k \not\equiv s_{k-1} \wedge send(s_k, < T, s, \sigma >, l_{k-1}) \ \mathbf{at_r} \ U_k \wedge U_k \in [V_{k-1} - \delta, V_{k-1} - \gamma]$

holds. Then we obtain $V_{k-1} \leq U_k + \delta$.

Together with (6), we obtain

$$V_1 \leq U_k + (k-2)(1+\rho)T_s + (k-1)\delta. \tag{7}$$

Since s_k is the initiator of σ, by $NSource(s_k, r)$, we have

$$initiate(s_k, \sigma) \ \mathbf{at_{s_k}} \ T \wedge s_k \equiv s \wedge U_k \in (T - \epsilon, T + T_s + \epsilon).$$

Together with (7), we obtain

$$V_1 < T + (k-1)((1+\rho)T_s + \delta) + \epsilon. \tag{8}$$

Combining (5) and (8), it results in $V < T + k((1+\rho)T_s + \delta) + \epsilon$.

Since $k \leq m$, we finally obtain $V < T + m((1+\rho)T_s + \delta) + \epsilon$. □

The following lemma shows that if p receives $< T, s, \sigma >$ at time V measured on the clock of a correct processor r and s is faulty, then any other correct processor q will receive $< T, s, \sigma >$ by time $V + d(p,q)((1+\rho)T_s + \delta)$ measured on the clock of r.

Lemma 6.4 (Correct Receiving)

$$receive(p, < T, s, \sigma >, l') \ \mathbf{at_r} \ V \wedge \neg correct(s) \wedge correct(q) \wedge p \not\equiv q \rightarrow$$
$$\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + d(p,q)((1+\rho)T_s + \delta)$$

Proof: Assume that the premise of the lemma holds. We prove this lemma by induction on the distance between p and q. Since $p \not\equiv q$, we start with $d(p,q) = 1$.

- $d(p,q) = 1$. By definition, p and q are connected by some correct link. Let l be that link. Then we have $link(l, p, q) \wedge correct(l)$. From $receive(p, < T, s, \sigma >, l') \ \mathbf{at_r} \ V$, by the only omission failure axiom 3.10, there exist a p_1 and a U_1 such that
  $p_1 \not\equiv p \wedge send(p_1, < T, s, \sigma >, l') \ \mathbf{at_r} \ U_1 \wedge U_1 \in [V - \delta, V - \gamma]$
  holds. Since $U_1 \leq V - \gamma$ and $\gamma > 0$, we have $V \geq U_1 + \gamma$ and then $V > U_1$. By the faulty sender lemma 6.2, we have $\neg correct(p_1)$. Thus correct processor q is not that sender p_1. By $Forward(p, r)$, p will send $< T, s, \sigma >$ to q along link l within $(1+\rho)T_s$ time units. Thus we have $send(p, < T, s, \sigma >, l) \ \mathbf{in_r} \ [V, V + (1+\rho)T_s]$. By definition, there exists an X such that $send(p, < T, s, \sigma >, l) \ \mathbf{at_r} \ X \wedge X \in [V, V + (1+\rho)T_s]$ holds.
  By the bounded communication axiom 3.5, we obtain
  $receive(q, < T, s, \sigma >, l) \ \mathbf{in_r} \ [X + \gamma, X + \delta]$.
  Together with $X \leq V + (1+\rho)T_s$, we have proved
  $\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + (1+\rho)T_s + \delta$, i.e.,
  $\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V + d(p,q)((1+\rho)T_s + \delta)$.

- $d(p,q) = k+1$ with $k \geq 1$. By definition, there must exist a processor q_1 and a link l_2 such that $correct(q_1) \wedge correct(l_2) \wedge link(l_2, q_1, q) \wedge d(p, q_1) = k \wedge d(q_1, q) = 1$ holds. By the induction hypothesis, we have $\exists l_1 : receive(q_1, < T, s, \sigma >, l_1) \ \mathbf{by_r} \ V + k((1+\rho)T_s + \delta)$.
  By definition, there exists a V_1 such that
  $\exists l_1 : receive(q_1, < T, s, \sigma >, l_1) \ \mathbf{at_r} \ V_1 \wedge V_1 \leq V + k((1+\rho)T_s + \delta)$.
  Since $correct(q)$ and $\neg correct(s)$ hold, we obtain $q \not\equiv s$.
  Then by the propagation lemma 5.2, we have
  $\exists l : receive(q, < T, s, \sigma >, l) \ \mathbf{by_r} \ V_1 + (1+\rho)T_s + \delta$, i.e.,

$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + (k+1)((1+\rho)T_s + \delta)$.
Therefore we have proved
$\exists l : receive(q, < T, s, \sigma >, l)$ $\mathbf{by_r}$ $V + d(p,q)((1+\rho)T_s + \delta)$. □

Next lemma shows that if correct processor p learns of $< T, s, \sigma >$, then any correct processor q also learns of $< T, s, \sigma >$.

Lemma 6.5 (All Learn)

$$Learn(p, < T, s, \sigma >) \wedge correct(p) \wedge correct(q) \rightarrow Learn(q, < T, s, \sigma >)$$

Proof: Assume that the premise of the lemma holds. By $Learn(p, < T, s, \sigma >)$, we have

$(initiate(p, \sigma)$ $\mathbf{at_p}$ $T \wedge p \equiv s) \vee$ (1)

$(\exists l_1, V_1 : receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_p}$ $V_1 \wedge p \not\equiv s)$ (2)

From (2), by the initiation lemma 6.1, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T.
Since either (1) or (2) holds, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T from the premise.
We have to prove $Learn(q, < T, s, \sigma >)$, i.e., the following formula:

$(initiate(q, \sigma)$ $\mathbf{at_q}$ $T \wedge q \equiv s) \vee$ (3)

$(\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $V_2 \wedge q \not\equiv s)$. (4)

There are two possibilities:

- if $s \equiv q$, then we have $initiate(q, \sigma)$ $\mathbf{at_q}$ $T \wedge q \equiv s$ holds, i.e., (3) holds;
- if $s \not\equiv q$, we prove that (4) holds by the following two cases.

  1. If $correct(s)$ holds, by the bounded receiving lemma 5.3, we obtain
     $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{by_s}$ $T + d(s,q)((1+\rho)T_s + \delta)$.
     By the clock synchronization lemma 3.1, we have
     $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{before_q}$ $T + d(s,q)((1+\rho)T_s + \delta) + \epsilon$, i.e.,
     $\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $V_2 \wedge q \not\equiv s$.
     Hence (4) holds.
  2. If $\neg correct(s)$ holds, since $correct(p)$ holds, we obtain $p \not\equiv s$ and then (1) does not hold. From (2), we have $receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_p}$ V_1. Then there exists a V_1' such that $receive(p, < T, s, \sigma >, l_1)$ $\mathbf{at_q}$ $V_1' \wedge V_1' \in (V_1 - \epsilon, V_1 + \epsilon)$. Hence there must exist a processor p_1 which is one of the first correct processors that have received $< T, s, \sigma >$ according to the clock of q. Thus there exist l_3 and V such that $Firstrec(p_1, < T, s, \sigma >, l_3)$ $\mathbf{at_q}$ V and hence $receive(p_1, < T, s, \sigma >, l_3)$ $\mathbf{at_q}$ V holds. By the first correct receiving lemma 6.3, we obtain $V < T + m((1+\rho)T_s + \delta) + \epsilon$.
     There are again two possibilities:
     - if $q \equiv p_1$, then we have $receive(q, < T, s, \sigma >, l_3)$ $\mathbf{at_q}$ V, i.e.,
       $\exists l_2, V_2 : (receive(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $V_2 \wedge V_2 < T + m((1+\rho)T_s + \delta) + \epsilon)$;
     - if $q \not\equiv p_1$, by the correct receiving lemma 6.4, we have
       $\exists l_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{by_q}$ $V + d(p,q)((1+\rho)T_s + \delta)$, i.e.,
       $\exists l_2, V_2 : (receive(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $V_2 \wedge$
       $V_2 < T + (d(p,q) + m)((1+\rho)T_s + \delta) + \epsilon)$.

     For both possibilities, we have
     $\exists l_2, V_2 : receive(q, < T, s, \sigma >, l_2)$ $\mathbf{at_q}$ $V_2 \wedge q \not\equiv s$, i.e., (4) holds. □

Next lemma expresses that if correct processor p conveys $< T, s, \sigma >$ at local time U, then any correct processor q conveys $< T, s, \sigma >$ in the interval $[T + T_r, T + T_r + T_c]$ on its own clock.

Lemma 6.6 (All Convey)

$$convey(p, < T, s, \sigma >) \ \mathbf{at_p} \ U \wedge correct(p) \wedge correct(q) \rightarrow$$
$$convey(q, < T, s, \sigma >) \ \mathbf{in_q} \ [T + T_r, T + T_r + T_c]$$

Proof: Assume that the premise of this lemma holds. From *correct*(p), we have *Origin*(p). By $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U, we obtain $Learn(p, < T, s, \sigma >)$. Then by the all learn lemma 6.5, we have $Learn(q, < T, s, \sigma >)$, i.e.,

$$(initiate(q, \sigma)\ \mathbf{at_q}\ T \wedge q \equiv s) \vee \quad (1)$$
$$(\exists l, V : receive(q, < T, s, \sigma >, l)\ \mathbf{at_q}\ V \wedge q \not\equiv s). \quad (2)$$

If (1) holds, by $Start(q)$, we have $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.
If (2) holds, by $Relay(q)$, we have $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.
Thus for both cases, we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$. □

Next we prove a theorem which shows that the atomicity property follows from the axioms and lemmas given before.

Theorem 6.1 (Atomicity) If $D_2 \geq T_c$, then

$$deliver(p, \sigma)\ \mathbf{at_p}\ U \wedge correct(p) \wedge correct(q) \rightarrow$$
$$\exists s, T : initiate(s, \sigma)\ \mathbf{at_s}\ T \wedge deliver(q, \sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2],$$

i.e., the atomicity property *ATOM* holds.

Proof: Assume that the premise of the theorem holds. From $deliver(p, \sigma)$ $\mathbf{at_p}$ U, by definition, there exist s and T such that $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U holds. By the server process specification axiom 4.1 and *correct*(p), we have *Origin*(p). By *Origin*(p), we obtain
$Learn(p, < T, s, \sigma >) \wedge U \in [T + T_r, T + T_r + T_c]$, i.e.,

$$((initiate(p, \sigma)\ \mathbf{at_p}\ T \wedge p \equiv s) \vee \quad (1)$$
$$(\exists l, V : receive(p, < T, s, \sigma >, l)\ \mathbf{at_p}\ V \wedge p \not\equiv s)) \wedge \quad (2)$$
$$U \in [T + T_r, T + T_r + T_c]. \quad (3)$$

From (1), we have $initiate(s, \sigma)$ $\mathbf{at_s}$ T.
From (2), by the initiation lemma 6.1, we obtain $initiate(s, \sigma)$ $\mathbf{at_s}$ T.
Thus for both cases, we have

$$\exists s, T : initiate(s, \sigma)\ \mathbf{at_s}\ T. \quad (4)$$

From $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U, by the all convey lemma 6.6, we have
$convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[T + T_r, T + T_r + T_c]$.
From (3), we have $T \in [U - T_r - T_c, U - T_r]$.
Hence we obtain $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ $[U - T_c, U + T_c]$.
By definition, we obtain $deliver(q, \sigma)$ $\mathbf{in_q}$ $[U - T_c, U + T_c]$.
Since $D_2 \geq T_c$, we have

$$deliver(q, \sigma)\ \mathbf{in_q}\ [U - D_2, U + D_2]. \quad (5)$$

From (4) and (5), this theorem holds. □

7 Verification of Order

The order property of the atomic broadcast protocol will be proved in this section. We first give two lemmas which will be used to prove the order property.

The following lemma shows that, for any correct processors p and q, if p conveys $< T, s, \sigma >$ at local time U, q conveys $< T, s, \sigma >$ at local time V, and no update is delivered by p in the interval $[0, U)$, then there is also no update delivered by q in the interval $[0, V)$.

Lemma 7.1 (First Delivery)

$$convey(p, < T, s, \sigma >)\ \mathbf{at_p}\ U \wedge convey(q, < T, s, \sigma >)\ \mathbf{at_q}\ V \wedge correct(p) \wedge correct(q) \wedge$$
$$\neg deliver(p)\ \mathbf{in_p}\ [0, U) \rightarrow \neg deliver(q)\ \mathbf{in_q}\ [0, V).$$

Proof: Assume that the premise of this lemma holds. Suppose $deliver(q)$ $\mathbf{in_q}$ $[0, V)$ holds. By definition, there exist s_0, T_0, and V_0 such that $convey(q, < T_0, s_0, \sigma_0 >)$ $\mathbf{at_q}$ $V_0 \wedge V_0 \in [0, V)$

holds. By assumption, we have $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ V.
From $V_0 < V$, by $Sequen(q)$, we obtain $(T_0, s_0) \sqsubset (T, s)$.
By the all convey lemma 6.6, we have $convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{in_p}$ $[T_0 + T_r, T_0 + T_r + T_c]$, i.e., there exists a $U_0 \in CVAL$ such that $convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{at_p}$ U_0 holds.
By assumption, we have $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U.
Since $(T_0, s_0) \sqsubset (T, s)$, by $Sequen(p)$, we obtain $U_0 < U$.
From $U_0 \in CVAL$, we have $U_0 \geq 0$ and thus $U_0 \in [0, U)$. Therefore we obtain $convey(p, < T_0, s_0, \sigma_0 >)$ $\mathbf{at_p}$ $U_0 \wedge U_0 \in [0, U)$, i.e., $deliver(p, \sigma_0)$ $\mathbf{in_p}$ $[0, U)$.
But by assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U)$. Thus it leads to contradiction and then $deliver(q)$ $\mathbf{in_q}$ $[0, V)$ does not hold, i.e., $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V)$ holds. □

Next lemma shows that, for any correct processors p and q, if p conveys $< T_1, s_1, \sigma_1 >$ at local time U_1 and $< T_2, s_2, \sigma_2 >$ at local time U_2, q conveys $< T_1, s_1, \sigma_1 >$ at local time V_1 and $< T_2, s_2, \sigma_2 >$ at local time V_2, and there is no update delivered by p in the interval (U_1, U_2), then there is also no update delivered by q in the interval (V_1, V_2).

Lemma 7.2 (No Delivery)

$$convey(p, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ U_1 \wedge convey(p, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ U_2 \wedge correct(p) \ \wedge$$
$$convey(q, < T_1, s_1, \sigma_1 >) \ \mathbf{at_p} \ V_1 \wedge convey(q, < T_2, s_2, \sigma_2 >) \ \mathbf{at_p} \ V_2 \wedge correct(q) \ \wedge$$
$$\neg deliver(p) \ \mathbf{in_p} \ (U_1, U_2) \rightarrow \neg deliver(q) \ \mathbf{in_q} \ (V_1, V_2).$$

Proof: Assume that the premise of this lemma holds. Suppose $deliver(q)$ $\mathbf{in_q}$ (V_1, V_2) holds. By definition, there exist s and T such that $convey(q, < T, s, \sigma >)$ $\mathbf{in_q}$ (V_1, V_2) holds. Then there exists a V such that $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ $V \wedge V \in (V_1, V_2)$ holds.
By assumption, we have $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ V_1.
Since $V_1 < V$, by $Sequen(q)$, we obtain $(T_1, s_1) \sqsubset (T, s)$.
Similarly, from assumption, we have $convey(q, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ V_2.
Since $V < V_2$, by $Sequen(q)$ again, we obtain $(T, s) \sqsubset (T_2, s_2)$.
From $convey(q, < T, s, \sigma >)$ $\mathbf{at_q}$ V, by the all convey lemma 6.6, we have $convey(p, < T, s, \sigma >)$ $\mathbf{in_p}$ $[T + T_r, T + T_r + T_c]$,
i.e., there exists a U such that $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ U holds.
By assumption, we have $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1.
Since $(T_1, s_1) \sqsubset (T, s)$, by $Sequen(p)$, we obtain $U_1 < U$.
Similarly, from assumption, we have $convey(p, < T_2, s_2, \sigma_2 >)$ $\mathbf{at_p}$ U_2.
Since $(T, s) \sqsubset (T_2, s_2)$, by $Sequen(p)$, we obtain $U < U_2$.
Thus we obtain $convey(p, < T, s, \sigma >)$ $\mathbf{at_p}$ $U \wedge U \in (U_1, U_2)$.
By definition, we have $deliver(p, \sigma)$ $\mathbf{in_p}$ (U_1, U_2).
But from assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ (U_1, U_2).
Thus it leads to contradiction and then $deliver(q, \sigma)$ $\mathbf{in_q}$ (V_1, V_2) does not hold,
i.e., $\neg deliver(q)$ $\mathbf{in_q}$ (V_1, V_2) holds. □

Next we prove, by the following theorem, that the order property holds.

Theorem 7.1 (Order)

$$correct(p) \wedge correct(q) \rightarrow \forall U \exists V : List(p, U) \subseteq List(q, V),$$

i.e., the order property holds.

Proof: For any clock value $U \in CVAL$, assume $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle \in List(p, U)$. We prove that there exists a V such that $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle \in List(q, V)$.
By definition, there exist $k \in I\!N^+$, $U_1, U_2, \ldots, U_k$ such that $U_1 \leq U_2 \leq \ldots \leq U_k < U$, $deliver(p, \sigma_i)$ $\mathbf{at_p}$ U_i, for $i = 1, 2, \ldots, k$, $\neg deliver(p)$ $\mathbf{in_p}$ (U_j, U_{j+1}), for $j = 1, 2, \ldots, k-1$, and $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$. From $deliver(p, \sigma_i)$ $\mathbf{at_p}$ U_i, there exist s_i and T_i such that

$convey(p, < T_i, s_i, \sigma_i >)$ $\mathbf{at_p}$ U_i holds. Let $V = U + T_c$. We show, by induction on k, that there exist $V_1, V_2, \ldots, V_k$ such that $V_1 \le V_2 \le \ldots \le V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k$, $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, 2, \ldots, k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.

- $k = 1$. By assumption, we have $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_p}$ U_1 and $\neg deliver(p)$ $\mathbf{in_p}$ $[0, U_1)$. By the all convey lemma 6.6, we obtain
  $convey(p, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_p}$ $[T_1 + T_r, T_1 + T_r + T_c]$ and
  $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{in_q}$ $[T_1 + T_r, T_1 + T_r + T_c]$.
  Thus we have $U_1 \in [T_1 + T_r, T_1 + T_r + T_c]$. Since $U_1 < U$, we obtain $T_1 + T_r < U$.
  Then there exists a $V_1 \in CVAL$ such that
  $convey(q, < T_1, s_1, \sigma_1 >)$ $\mathbf{at_q}$ $V_1 \wedge V_1 \in [T_1 + T_r, T_1 + T_r + T_c]$ holds.
  Thus we have $V_1 \le T_1 + T_r + T_c$ and hence $V_1 < U + T_c$, i.e., $V_1 < V$.
  By the first deliver lemma 7.1, we also obtain $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.

- $k > 1$. By the induction hypothesis, there exist $V_1, V_2, \ldots, V_{k-1}$ such that $V_1 \le V_2 \le \ldots \le V_{k-1}$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k-1$,
  $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, 2, \ldots, k-2$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.
  By assumption, we have $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
  By the all convey lemma 6.6, there exists a V_k such that
  $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ $V_k \wedge V_k \in [T_k + T_r, T_k + T_r + T_c]$ holds.
  Since $U_{k-1} \le U_k$, we prove $V_{k-1} \le V_k$ by the following two cases.

  1. Assume $U_{k-1} < U_k$. By assumption, we have
     $convey(p, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_p}$ U_{k-1} and $convey(p, < T_k, s_k, \sigma_k >)$ $\mathbf{at_p}$ U_k.
     Since $U_{k-1} < U_k$, by $Sequen(p)$, we obtain $(T_{k-1}, s_{k-1}) \sqsubset (T_k, s_k)$.
     From the induction hypothesis and above, we have
     $convey(q, < T_{k-1}, s_{k-1}, \sigma_{k-1} >)$ $\mathbf{at_q}$ V_{k-1} and $convey(q, < T_k, s_k, \sigma_k >)$ $\mathbf{at_q}$ V_k.
     Since $(T_{k-1}, s_{k-1}) \sqsubset (T_k, s_k)$, by $Sequen(q)$, we obtain $V_{k-1} < V_k$.
  2. Assume $U_{k-1} = U_k$.
     Suppose $V_{k-1} < V_k$. Similar as above, we obtain $U_{k-1} < U_k$ which does not hold.
     Suppose $V_{k-1} > V_k$. Similarly, we obtain $U_{k-1} > U_k$ which also does not hold.
     Therefore only $V_{k-1} = V_k$ holds.

  Combining these two cases, we obtain $V_{k-1} \le V_k$.
  Similar to the case for $k = 1$, we have $U_k \in [T_k + T_r, T_k + T_r + T_c]$ and $U_k < U$. Thus we obtain $T_k + T_r < U$. Since $V_k \le T_k + T_r + T_c$, we have $V_k < U + T_c$, i.e., $V_k < V$.
  By assumption, we have $\neg deliver(p)$ $\mathbf{in_p}$ (U_{k-1}, U_k).
  Then by the no delivery lemma 7.2, we obtain $\neg deliver(q)$ $\mathbf{in_q}$ (V_{k-1}, V_k).

Hence we have proved that there exist $V_1, V_2, \ldots, V_k$ such that $V_1 \le V_2 \le \ldots \le V_k < V$, $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k$, $\neg deliver(q)$ $\mathbf{in_q}$ (V_j, V_{j+1}), for $j = 1, 2, \ldots, k-1$, and $\neg deliver(q)$ $\mathbf{in_q}$ $[0, V_1)$.
Since $convey(q, < T_i, s_i, \sigma_i >)$ $\mathbf{at_q}$ V_i implies $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, we obtain $deliver(q, \sigma_i)$ $\mathbf{at_q}$ V_i, for $i = 1, 2, \ldots, k$. Therefore $\langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle \in List(q, V)$.
Hence for any U there exists a V, i.e., $V = U + T_c$, such that $List(p, U) \subseteq List(q, V)$. $\square$

8 Comparison and Conclusion

We have formally proved that the termination, atomicity, and order properties of the protocol hold, provided

1. $D_1 \geq T_r + T_c$, where D_1 is the broadcast termination time in the termination property specification, T_r is the time to ensure that all correct processors have received a message containing an update after it is initiated, and T_c is the time for a correct processor to convey updates to its client processes;

2. $D_2 \geq T_c$, where D_2 is the difference of delivery time of an update by two correct processors in the atomicity property specification;

3. $T_r \geq T_s \geq 0$, $T_c \geq 0$, $\delta \geq \gamma > 0$, $\epsilon > 0$, and $\rho \geq 0$, where T_s is the time for a correct processor to send a message to its neighbors, γ and δ are the lower and upper bounds, respecitively, of message transmission delay between two correct processors, ϵ and ρ are the maximal deviation and speed difference, respectively, of local clocks of correct processors.

Comparing our paper with [CASD89], the basic ideas of proving properties of the protocol are similar. In the algorithm for the protocol in that paper, a processor only relays a message to its neighbors if the message is received by the processor for the first time and it is not a "late message". Actually these two factors do not affect the correctness of the protocol. Adding them to the algorithm is to improve the efficiency of the implementation. Thus the informal proof in that paper verifies the protocol without taking these factors into account. We did the formal proof similarly and this can be seen from the $Relay(p)$ property.

From the first correct receiving lemma 6.3 and the correct receiving lemma 6.4, we observe that if an update σ is initiated by a processor s at local clock time T, then any correct provessor p will receive the message $< T, s, \sigma >$ before $(d+m)((1+\rho)T_s + \delta) + \epsilon$ measured on its own clock, where d is the maximal distance between two correct processors and m is the maximal number of faulty processors. Thus $T_r \geq (d+m)((1+\rho)T_s + \delta) + \epsilon$. The corresponding time in [CASD89] is $(d+m)\delta + \epsilon$. If we assume $T_s = 0$ and $\rho = 0$ as in [CASD89], then we obtain the same bound. Notice that the condition on T_r is only needed for the implementation of the server specification $Spec(p)$, not directly for the correctness proof of the protocol.

In [CASD89], clock synchronization is assumed for always correct processors. To give a precise proof of the protocol, e.g. in the proof of lemma 5.1, we needed a more refined clock synchronization assumtion for processors which are correct at some time points. Thus we took this assumption as an axiom and the assumption in [CASD89] as a lemma.

To prove the atomicity property, we need to show that if a correct processor p delivers σ at some time U, then σ was initiated by some processor s at some clock time T. This is not proved in [CASD89]. We have proved it in lemma 6.1 by using available timing information. There we needed a lower bound for message transmission delay between two correct processors. Thus we add a lower bound γ in the bounded communication axiom 3.5.

There is an implicit assumption in [CASD89] about the drift of local clocks. We have formalized this assumption in axiom 3.8. This axiom is used in lemma 5.1 to formulate part of the server specification in terms of the local clock of any correct processor. Together with the other axioms about local clocks, i.e., the synchronization axiom 3.6 and the monotonic clock axiom 3.7, this makes it possible to perform the verification in terms of local clock values, similar to the informal reasoning in [CASD89]. In contrast with most formal methods, see e.g. [BHRR91], there is no need to refer to global times during the protocol verification. This leads to a convenient and natural calculus.

There is quickly growing literature on the formal verification of real-time and fault-tolerant distributed systems. Closely related to our approach is the recent work on the proof checker EHDM and its successor PVS. Rushby and von Henke [RH93] use EHDM to check the proofs of Lamport and Melliar-Smith's interactive convergence clock synchronization algorithm [LMS85]. Mechanical verification of a generalized protocol for Byzantine fault-tolerant clock synchronization [Sch87] by using EHDM is described in [Sha92]. In future applications of our approach we will certainly investigate the use of such an interactive proof checker.

Observe that the formal method used in our paper is compositional. It enables us to reason with only specifications and abstract from the implementation details. A natural continuation of this work is to implement the server specification and verify that it is indeed a correct implementation.

Acknowledgements:

Many thanks go to Flaviu Cristian for his interest and valuable comments on the draft of this paper.

References

[BD85] O. Babaoglu and R. Drumond. Streets of byzantium: Network architectures for fast reliable broadcast. *IEEE Transactions on Software Engineering 11(6)*, 1985.

[BHRR91] J.W. de Bakker, C. Huizing, W.-P. de Roever, and G. Rozenberg(Eds.). *Real-Time: Theory in Practice.* LNCS 600, Springer-Verlag, 1991.

[BJ87] K. Birman and T. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems 5(1)*, pages 47–76, 1987.

[CAS86] F. Cristian, H. Aghili, and R. Strong. Approximate clock synchronization despite omission and performance failures and processor joins. In *The 16th International Symposium on Fault-Tolerant Computing.* Wien, Austrian, 1986.

[CAS93] F. Cristian, H. Aghili, and R. Strong. Clock synchronization in the presence of omission and performance failures, and processor joins. In *Global States and Time in Distributed Systems.* Z. Yang and T.A. Marsland (Eds.), IEEE Computer Society Press, 1993.

[CASD85] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. In *The 15th Annual International Symposium on Fault-Tolerant Computing*, pages 200 – 206. Ann Arbor, USA, 1985.

[CASD89] F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. Research Report RJ 5244, IBM Almaden Research Center, 1989.

[CM84] J.M. Chang and N. Maxemchuck. Reliable broadcast protocols. *ACM Transactions on Computer Systems 2(3)*, pages 251–273, 1984.

[Cri90] F. Cristian. Synchronous atomic broadcast for redundant broadcast channels. *The Journal of Real-Time Systems 2*, pages 195–212, 1990.

[Cri93] F. Cristian. Comments. *Private Correspondence*, 1993.

[Hoo91] J. Hooman. *Specification and Compositional Verification of Real-Time Systems.* LNCS 558, Springer-Verlag, 1991.

[LMS85] L. Lamport and P.M. Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[RH93] J. Rushby and F. von Henke. Formal verification of algorithms for critical systems. *IEEE Transactions on Software Engineering*, 19(1):13–23, 1993.

[Sch87] F.B. Schneider. Understanding protocols for Byzantine clock synchronization. Technical report 87-859, Dept. of Computer Science, Cornell University, 1987.

[Sha92] N. Shankar. Mechanical verification of a generalized protocol for Byzantine fault tolerant clock synchronization. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 217–236. J. Vytopil(Ed.), LNCS 571, Springer-Verlag, 1992.

[ZH92] P. Zhou and J. Hooman. A proof theory for asynchronously communicating real-time systems. In *Proc. of the 13th IEEE Real-Time Systems*, pages 177–186. IEEE Computer Society Press, 1992.

[Zwi89] J. Zwiers. *Compositionality, Concurrency and Partial Correctness.* LNCS 321, Springer-Verlag, 1989.

In this series appeared:

91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p. 49.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.
91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.

91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Some categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. 25.
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.
91/29	H. Ten Eikelder R. van Geldrop	Correctness of Acceptor Schemes for Regular Languages, p. 31.
91/30	J.C.M. Baeten F.W. Vaandrager	An Algebra for Process Creation, p. 29.
91/31	H. ten Eikelder	Some algorithms to decide the equivalence of recursive types, p. 26.
91/32	P. Struik	Techniques for designing efficient parallel programs, p. 14.
91/33	W. v.d. Aalst	The modelling and analysis of queueing systems with QNM-ExSpect, p. 23.
91/34	J. Coenen	Specifying fault tolerant programs in deontic logic, p. 15.

91/35	F.S. de Boer J.W. Klop C. Palamidessi	Asynchronous communication in process algebra, p. 20.
92/01	J. Coenen J. Zwiers W.-P. de Roever	A note on compositional refinement, p. 27.
92/02	J. Coenen J. Hooman	A compositional semantics for fault tolerant real-time systems, p. 18.
92/03	J.C.M. Baeten J.A. Bergstra	Real space process algebra, p. 42.
92/04	J.P.H.W.v.d.Eijnde	Program derivation in acyclic graphs and related problems, p. 90.
92/05	J.P.H.W.v.d.Eijnde	Conservative fixpoint functions on a graph, p. 25.
92/06	J.C.M. Baeten J.A. Bergstra	Discrete time process algebra, p.45.
92/07	R.P. Nederpelt	The fine-structure of lambda calculus, p. 110.
92/08	R.P. Nederpelt F. Kamareddine	On stepwise explicit substitution, p. 30.
92/09	R.C. Backhouse	Calculating the Warshall/Floyd path algorithm, p. 14.
92/10	P.M.P. Rambags	Composition and decomposition in a CPN model, p. 55.
92/11	R.C. Backhouse J.S.C.P.v.d.Woude	Demonic operators and monotype factors, p. 29.
92/12	F. Kamareddine	Set theory and nominalisation, Part I, p.26.
92/13	F. Kamareddine	Set theory and nominalisation, Part II, p.22.
92/14	J.C.M. Baeten	The total order assumption, p. 10.
92/15	F. Kamareddine	A system at the cross-roads of functional and logic programming, p.36.
92/16	R.R. Seljée	Integrity checking in deductive databases; an exposition, p.32.
92/17	W.M.P. van der Aalst	Interval timed coloured Petri nets and their analysis, p. 20.
92/18	R.Nederpelt F. Kamareddine	A unified approach to Type Theory through a refined lambda-calculus, p. 30.
92/19	J.C.M.Baeten J.A.Bergstra S.A.Smolka	Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, p. 36.
92/20	F.Kamareddine	Are Types for Natural Language? P. 32.

92/21	F.Kamareddine	Non well-foundedness and type freeness can unify the interpretation of functional application, p. 16.
92/22	R. Nederpelt F.Kamareddine	A useful lambda notation, p. 17.
92/23	F.Kamareddine E.Klein	Nominalization, Predication and Type Containment, p. 40.
92/24	M.Codish D.Dams Eyal Yardeni	Bottum-up Abstract Interpretation of Logic Programs, p. 33.
92/25	E.Poll	A Programming Logic for Fω, p. 15.
92/26	T.H.W.Beelen W.J.J.Stut P.A.C.Verkoulen	A modelling method using MOVIE and SimCon/ExSpect, p. 15.
92/27	B. Watson G. Zwaan	A taxonomy of keyword pattern matching algorithms, p. 50.
93/01	R. van Geldrop	Deriving the Aho-Corasick algorithms: a case study into the synergy of programming methods, p. 36.
93/02	T. Verhoeff	A continuous version of the Prisoner's Dilemma, p. 17
93/03	T. Verhoeff	Quicksort for linked lists, p. 8.
93/04	E.H.L. Aarts J.H.M. Korst P.J. Zwietering	Deterministic and randomized local search, p. 78.
93/05	J.C.M. Baeten C. Verhoef	A congruence theorem for structured operational semantics with predicates, p. 18.
93/06	J.P. Veltkamp	On the unavoidability of metastable behaviour, p. 29
93/07	P.D. Moerland	Exercises in Multiprogramming, p. 97
93/08	J. Verhoosel	A Formal Deterministic Scheduling Model for Hard Real-Time Executions in DEDOS, p. 32.
93/09	K.M. van Hee	Systems Engineering: a Formal Approach Part I: System Concepts, p. 72.
93/10	K.M. van Hee	Systems Engineering: a Formal Approach Part II: Frameworks, p. 44.
93/11	K.M. van Hee	Systems Engineering: a Formal Approach Part III: Modeling Methods, p. 101.
93/12	K.M. van Hee	Systems Engineering: a Formal Approach Part IV: Analysis Methods, p. 63.
93/13	K.M. van Hee	Systems Engineering: a Formal Approach Part V: Specification Language, p. 89.

93/14	J.C.M. Baeten J.A. Bergstra	On Sequential Composition, Action Prefixes and Process Prefix, p. 21.
93/15	J.C.M. Baeten J.A. Bergstra R.N. Bol	A Real-Time Process Logic, p. 31.
93/16	H. Schepers J. Hooman	A Trace-Based Compositional Proof Theory for Fault Tolerant Distributed Systems, p. 27
93/17	D. Alstein P. van der Stok	Hard Real-Time Reliable Multicast in the DEDOS system, p. 19.
93/18	C. Verhoef	A congruence theorem for structured operational semantics with predicates and negative premises, p. 22.
93/19	G-J. Houben	The Design of an Online Help Facility for ExSpect, p.21.
93/20	F.S. de Boer	A Process Algebra of Concurrent Constraint Programming, p. 15.
93/21	M. Codish D. Dams G. Filé M. Bruynooghe	Freeness Analysis for Logic Programs - And Correctness?, p. 24.
93/22	E. Poll	A Typechecker for Bijective Pure Type Systems, p. 28.
93/23	E. de Kogel	Relational Algebra and Equational Proofs, p. 23.
93/24	E. Poll and Paula Severi	Pure Type Systems with Definitions, p. 38.
93/25	H. Schepers and R. Gerth	A Compositional Proof Theory for Fault Tolerant Real-Time Distributed Systems, p. 31.
93/26	W.M.P. van der Aalst	Multi-dimensional Petri nets, p. 25.
93/27	T. Kloks and D. Kratsch	Finding all minimal separators of a graph, p. 11.
93/28	F. Kamareddine and R. Nederpelt	A Semantics for a fine λ-calculus with de Bruijn indices, p. 49.
93/29	R. Post and P. De Bra	GOLD, a Graph Oriented Language for Databases, p. 42.
93/30	J. Deogun T. Kloks D. Kratsch H. Müller	On Vertex Ranking for Permutation and Other Graphs, p. 11.
93/31	W. Körver	Derivation of delay insensitive and speed independent CMOS circuits, using directed commands and production rule sets, p. 40.
93/32	H. ten Eikelder and H. van Geldrop	On the Correctness of some Algorithms to generate Finite Automata for Regular Expressions, p. 17.

93/33	L. Loyens and J. Moonen	ILIAS, a sequential language for parallel matrix computations, p. 20.
93/34	J.C.M. Baeten and J.A. Bergstra	Real Time Process Algebra with Infinitesimals, p.39.
93/35	W. Ferrer and P. Severi	Abstract Reduction and Topology, p. 28.
93/36	J.C.M. Baeten and J.A. Bergstra	Non Interleaving Process Algebra, p. 17.
93/37	J. Brunekreef J-P. Katoen R. Koymans S. Mauw	Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks, p. 73.
93/38	C. Verhoef	A general conservative extension theorem in process algebra, p. 17.
93/39	W.P.M. Nuijten E.H.L. Aarts D.A.A. van Erp Taalman Kip K.M. van Hee	Job Shop Scheduling by Constraint Satisfaction, p. 22.
93/40	P.D.V. van der Stok M.M.M.P.J. Claessen D. Alstein	A Hierarchical Membership Protocol for Synchronous Distributed Systems, p. 43.
93/41	A. Bijlsma	Temporal operators viewed as predicate transformers, p. 11.
93/42	P.M.P. Rambags	Automatic Verification of Regular Protocols in P/T Nets, p. 23.
93/43	B.W. Watson	A taxomomy of finite automata construction algorithms, p. 87.
93/44	B.W. Watson	A taxonomy of finite automata minimization algorithms, p. 23.
93/45	E.J. Luit J.M.M. Martin	A precise clock synchronization protocol,p.
93/46	T. Kloks D. Kratsch J. Spinrad	Treewidth and Patwidth of Cocomparability graphs of Bounded Dimension, p. 14.
93/47	W. v.d. Aalst P. De Bra G.J. Houben Y. Kornatzky	Browsing Semantics in the "Tower" Model, p. 19.
93/48	R. Gerth	Verifying Sequentially Consistent Memory using Interface Refinement, p. 20.

94/01	P. America M. van der Kammen R.P. Nederpelt O.S. van Roosmalen H.C.M. de Swart	The object-oriented paradigm, p. 28.
94/02	F. Kamareddine R.P. Nederpelt	Canonical typing and Π-conversion, p. 51.
94/03	L.B. Hartman K.M. van Hee	Application of Marcov Decision Processe to Search Problems, p. 21.
94/04	J.C.M. Baeten J.A. Bergstra	Graph Isomorphism Models for Non Interleaving Process Algebra, p. 18.