On the Formal Specification of Group Membership Services*

Emmanuelle Anceaume† Bernadette Charron-Bost‡
Pascale Minet§ Sam Toueg¶

Abstract

The problem of group membership has been the focus of much theoretical and experimental work on fault-tolerant distributed systems. This has resulted in a voluminous literature and several formal specifications of this problem have been given. In this paper, we examine the two most referenced formal specifications of group membership and show that they are unsatisfactory: One has flaws in the formalism and allows undesirable executions, and the other can be satisfied by useless protocols.

1 Introduction

Group membership is an important component of several experimental or commercial fault-tolerant distributed systems such as the *Highly Available System* [Cri87], *Isis* [Bir93], *Horus* [vRBC+93], *Transis* [ADKM92a], *Amoeba* [KT91], *Newtop* [EMS95], and *Relacs* [BDGB94]. Roughly speaking, a group membership protocol manages the formation and maintenance of a set of processes called a *group*. For example, a group may be a set of processes that are cooperating towards a common task (e.g., the primary and backup servers of a database), a set of processes that share a common interest (e.g., clients that subscribe to a particular newsgroup), or the set of all processes in the system that are currently deemed to be operational. Since new processes may want to join an existing group, and others may want to leave or have to be removed after they fail, the membership of a group changes dynamically. A group membership protocol must manage these changes in a coherent way: each process has a *local view* of the current membership of the group, and processes maintain some form of agreement on these local views.

*Research partially supported by NSF grant CCR-9402894 and DARPA/NASA Ames grant NAG-2-593

†INRIA, B.P. 105, 78153 Le Chesnay Cedex, FRANCE.

‡Laboratoire d'Informatique LIX, Ecole Polytechnique, 91128 Palaiseau Cedex, FRANCE.

§INRIA, B.P. 105, 78153 Le Chesnay Cedex, FRANCE.

¶Department of Computer Science, Upson Hall, Cornell University, Ithaca NY 14853, USA. This work was done while visiting the Reflecs Project at INRIA - Rocquencourt.

The group membership problem was first defined for synchronous systems by [Cri91]. Since then, this problem has also been the subject of intense investigation for asynchronous systems. In particular, two types of group membership services have emerged: *primary-partition*, e.g. [RB91, KT91, MPS91, MSMA94, HS95], and *partitionable*, e.g. [ADKM92b, ADKM92a, JFR93, vRBC$^+$93, BDGB94, EMS95].

Roughly speaking, a primary-partition group membership service maintains a *single* agreed view of the group (i.e., processes agree on their local views of the group). Such services are intended for systems with no network partitions, or for systems that allow the group membership to change in at most one network partition, the *primary partition*.

In contrast, a partitionable group membership service allows *multiple* views of the group to co-exist and evolve concurrently: There may be several disjoint subsets of processes such that processes in each subset agree that they are the current members of the group. In other words, such group membership services allow group splitting (e.g., when the network partitions) and group merging (e.g., when communication between partitions is restored).

Two papers are widely referenced as the first to give rigorous definitions of group membership for asynchronous systems: [RB91] for the primary-partition type, and [DMS94] for the partitionable one. Indeed, giving a formal specification of group membership is the principal goal of [RB91, DMS94] and their updated versions [Ric93, RB94, DMS95].[1] In this paper we show that these formal specifications are unsatisfactory: The primary-partition one has flaws in the formalism and allows undesirable executions, and the partitionable one can be satisfied by useless protocols. Thus, contrary to a widely-held belief, these specifications do not provide a rigorous foundation for the study of group membership.

2 Primary-Partition Group Membership Service

The first and most referenced work on primary-partition group membership service for asynchronous systems is the one in [RB91]. Its goal is to formally define this service (called *Strong Group Membership Problem* and denoted *S-GMP*) and to give a matching protocol. This work was continued in [Ric93], and later in [RB94]. We focus on [RB94], as it is the most up-to-date version of S-GMP at the time of writing.

In our study of this work, we found several deficiencies with the formal specification of S-GMP: there are flaws in the formalism, the formal specification allows undesirable executions (violation of safety), and the liveness requirement of the specification of S-GMP is not satisfied by the protocol given as a solution to S-GMP. In the following sections, we discuss each one of these problems in turn.

[1] The abstract of [RB91] states that "We present a rigorous, formal specification for group membership...then a solution for this problem...", and the introduction of [DMS94] says that "The contribution of this paper is in giving a formal specification of the membership problem allowing partitions." [DMS94] also refers to [RB91] as the "the first formal definition of the requirements of membership in asynchronous environments."

2.1 Flaws in the Formalism

Some formal definitions are incomplete, do not match the informal English description that precedes these definitions, or impose unreasonable requirements. We illustrate some of these problems below.

The specification of S-GMP is given in terms of five formal properties, GMP-0 to GMP-4, that must hold at all *consistent cuts* [CL85, Mat88]. These formal properties are expressed in terms of temporal logic formulas that contain predicates on consistent cuts (branching time logic is adopted). Unfortunately, the definition of some of these predicates is not complete: For some consistent cuts their values are not specified. This prevents a precise and unambiguous interpretation of the formal specification.

One example of a predicate whose definition is incomplete is $\text{IN-LOCAL}_p^x(q)$; this predicate is used in property GMP-3 of the formal specification. Roughly speaking, $\text{IN-LOCAL}_p^x(q)$ should indicate whether process p's x^{th} local view of the group, denoted $\mathsf{LocalView}_p^x$, contains process q. A problem arises when one needs to evaluate this predicate at a consistent cut c where p has *not yet* formed its x^{th} local view of the group: The paper does not say what the value of $\text{IN-LOCAL}_p^x(q)$ should be at such a cut c.

Unfortunately, this incomplete definition is not easy to rectify. At cut c, p has not yet formed $\mathsf{LocalView}_p^x$, and one cannot predict whether q will be contained in $\mathsf{LocalView}_p^x$ "in the future" (when this view will be formed) because this future is not uniquely determined. In fact, at this point in the computation there are three types of possible extensions of c: Those in which p forms an x^{th} local view that contains q, those in which p forms an x^{th} local view that does not contain q, and those in which p never forms an x^{th} local view (for example, p crashes before doing so).

Since it is impossible to determine the future value of $\text{IN-LOCAL}_p^x(q)$, the alternative is to select an arbitrary value, and assign it by convention to $\text{IN-LOCAL}_p^x(q)$ at all cuts c where $\mathsf{LocalView}_p^x$ is not yet defined. But doing so also raises major problems. In the Appendix we show why one cannot arbitrarily set $\text{IN-LOCAL}_p^x(q)$ to *true* or $false$ at a cut c where $\mathsf{LocalView}_p^x$ is not yet defined. The other possibility is to assign the value *undefined* to $\text{IN-LOCAL}_p^x(q)$ at c. But how do we interpret a temporal logic formula such as GMP-3 where some terms like $\text{IN-LOCAL}_p^x(q)$ can now take the value *undefined*? This interpretation requires a precise three-valued temporal logic framework, where every temporal and boolean operator has been redefined to account for the possibility of undefined terms. To the best of our knowledge, no such temporal logic exists in the literature.

Predicate $\text{IN-LOCAL}_p^x(q)$ is not the only predicate whose definition is incomplete: The same problem exists with predicates IN-GP_p and OUT-GP_p which are used to formally express GMP-4, the liveness property of the specification. In fact, [RB94] states "The formula IN-GP_p holds on c exactly when p is a member of the group view at c (provided it exists), while OUT-GP_p holds when p is not a member (also provided the group view exists at c)". This definition does not specify the value of these predicates at cuts c such that the group view at c does *not* exist (it is clear that such cuts are possible).

We now give two examples of unreasonable requirements that are enforced by the formal specification of S-GMP. The first one concerns GMP-3, and the second one GMP-4. To explain the problem with GMP-3, we need to introduce some notation from [RB94]. Predicate $\textsc{down}_p$ holds at a cut c "exactly when p has crashed in c". The local membership view of process q on cut c is denoted $\mathsf{LocalView}_q(c)$, and $\mathsf{LocalView}_q$ is used "when the cut is clear from the context." GMP-3 requires that at all consistent cuts c the following holds:

$$c \models \bigwedge_{0\leq x} \bigwedge_{p} \Diamond \bigwedge_{q} \left(\textsc{in-local}_p^x(q) \Rightarrow \textsc{down}_q \vee (\mathsf{LocalView}_q = \mathsf{LocalView}_p^x)\right) \bigwedge$$
$$(\ (\ \neg\textsc{in-local}_p^x(q) \wedge \bigvee_{y<x} \textsc{in-local}_p^y(q)\ ) \Rightarrow \Box\textsc{notdef'd}(\mathsf{LocalView}_q^x)\ ).$$

Since $\Diamond(\phi \wedge \psi)$ implies $\Diamond\phi \wedge \Diamond\psi$:

$$c \models \bigwedge_{0\leq x} \bigwedge_{p} \bigwedge_{q} \Diamond \left(\textsc{in-local}_p^x(q) \Rightarrow \textsc{down}_q \vee (\mathsf{LocalView}_q = \mathsf{LocalView}_p^x)\right).$$

Pick a process p, and focus on the predicate corresponding to $x = 0$ and $q = p$ in the above conjunction:

$$c \models \Diamond \left(\textsc{in-local}_p^0(p) \Rightarrow \textsc{down}_p \vee (\mathsf{LocalView}_p = \mathsf{LocalView}_p^0)\right).$$

Consider p's initial local view $\mathsf{LocalView}_p^0$. The paper states that p is always in its local view, thus $p \in \mathsf{LocalView}_p^0$ and $\textsc{in-local}_p^0(p)$ holds. This implies:

$$c \models \Diamond \left(\textsc{down}_p \vee (\mathsf{LocalView}_p = \mathsf{LocalView}_p^0)\right).$$

Consider a cut c such that p has already done several changes to its local view of the group. The above requires that in *all* extensions of c, either p crashes or p is forced to re-install its initial view of the group — a requirement that does not make sense.

The formal requirement GMP-4 is also unreasonable. Suppose that at some point a process p (that is in the group view) suspects that q is faulty. The English explanation of GMP-4 says that in this case "eventually either p is removed from the group view, or q is removed from it..." This should be translated by $\Diamond(\textsc{out-gp}_q \vee \textsc{out-gp}_p)$, rather than by $\Diamond\textsc{out-gp}_q \vee \Diamond\textsc{out-gp}_p$ as written in the paper. These two formulas are not equivalent: The branching-time logic interpretation of $\Diamond(\textsc{out-gp}_q \vee \textsc{out-gp}_p)$ at a cut c is that in every extension of c, either p or q is removed — a credible requirement. In contrast, the interpretation of $\Diamond\textsc{out-gp}_q \vee \Diamond\textsc{out-gp}_p$ is that in every extension of c process q is removed, or in every extension of c process p is removed. In other words, this requirement enforces a "uniform future" independent of the future behavior of the system. This is unreasonable because, as [RB91] noted, "The outcome [of whether it is p or q that is actually removed] will depend on the pattern of communication that ensues." This outcome also depends on the behavior of p and q after the cut c.

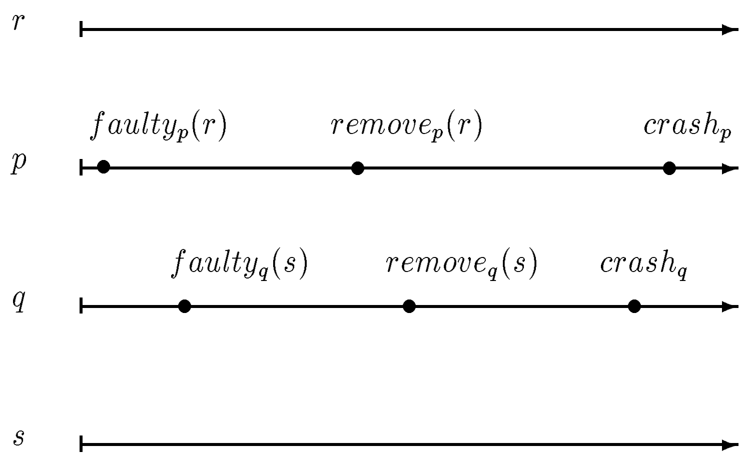

Figure 1: PROCESSES p AND q DISAGREE ON VIEW $V^{(1)}$

2.2 Violation of Safety

A primary-partition group membership service requires that the group views form a unique sequence such that there is no disagreement on the membership of the x^{th} view of the group (for every x). Indeed, the informal description of GMP-3 (one of the two "safety" properties in the formal specification of S-GMP) explains that "All processes, while they are member of core and provided they are not crashed, exhibit the same sequence of local views". The formal specification of S-GMP given in [RB91, Ric93, RB94], however, does not enforce this crucial requirement. In fact, we found that the formal specification allows an undesirable execution where *two processes disagree on the second view of the group.*

The scenario which is illustrated in Figure 2.2 depicts a system of four processes r, p, q, and s. Initially, all processes have the same local view $V^{(0)} = \{r, p, q, s\}$ of the group. Processes r and s "do nothing". Process p first suspects that r is faulty (i.e., p executes event $faulty_p(r)$ in the formalism of [RB94]), and then it removes r from its local view of the group (i.e., p executes event $remove_p(r)$). So p's local view of the group becomes $V_p^{(1)} = \{p, q, s\}$. An arbitrarily long time later, p crashes. The execution of process q is symmetric with regard to process s, and q's local view of the group becomes $V_q^{(1)} = \{r, p, q\}$. An arbitrarily long time later, q crashes.

In this scenario, for an arbitrarily long time (i.e., from the time p and q change their local views, to the time they crash) *p and q disagree on what view $V^{(1)}$ is.* In other words, they disagree on the membership of the second view of the group — two versions of $V^{(1)}$ exist concurrently.

It turns out that this scenario satisfies all the properties of the formal specification of S-GMP given in [RB94] (including the two properties called "Uniqueness" and "Sequence" that were intended to prevent such undesirable executions), but it clearly violates the semantics of a primary-partition group membership that this formal specification tries to capture.

2.3 Violation of Liveness

As we already mentioned in Section 2.1, the temporal logic definition of GMP-4 (the "liveness" property of S-GMP) is problematic because it is expressed in terms of predicates some of which are not defined in all cases. Thus, we must rely on the English definition of GMP-4: if a process p suspects that a process q is faulty (and p is in the group view) then eventually either p is removed from the group view, or q is removed from it.

As we explain below, the protocol given as a solution to S-GMP in asynchronous systems violates GMP-4, and so it does not satisfy the formal specification of S-GMP. In other words, the so-called "S-GMP protocol" does not quite solve the S-GMP problem. This contradicts several statements in [RB91, Ric93, RB94]. For example, [Ric93, RB94] state that the S-GMP protocol "correctly solves Strong GMP", and [Ric93] proves a theorem stating that the "S-GMP [protocol] satisfies GMP-4" (Theorem 5.2.5).[2]

We now describe an execution of the S-GMP protocol that violates the liveness property GMP-4. In this protocol, a process denoted mgr is in charge of coordinating updates of the group view. Suppose mgr believes that a process q in the group view is faulty. Property GMP-4 requires that either mgr or q is eventually removed from the group view. We now describe a protocol execution in which this does not occur. According to the S-GMP protocol, mgr submits a view change to the members of its current group view. Now suppose that while waiting for acknowledgements, mgr comes to believe that a majority of processes in its current group view are faulty. The protocol then forces mgr to crash. The following can now happen: every other process successively takes over the task of coordinating the installation of the new group view, but then believes that a majority of processes are faulty, and thus crashes itself. In this execution of the S-GMP protocol, there is a *collective suicide* and no view change occurs. Thus neither q nor mgr are removed from the existing group view — a violation of GMP-4. Note that this collective suicide can occur even if there are no process or communication failures.

3 Partitionable Group Membership Service

Recall that a primary-partition group membership service is supposed to maintain a *single agreed view* of the current membership of a group. In contrast to primary-partition group membership services, *partitionable* ones allow processes to disagree on the current membership of the group, i.e., *several different views* of the membership of the group

[2] On the other hand, the authors seem to be aware of the fact that the S-GMP protocol does not satisfy the liveness requirement of the S-GMP specification. In fact, "Guaranteeing progress in all possible executions" is listed as one of the "Non-Goals" in Figure 1 of [RB94]. But this "non-goal" is in apparent contradiction with the liveness property GMP-4.

may evolve concurrently and independently from each other [DMS94, DMS95, JFR93, vRBC+93, BDGB94, EMS95]. In particular, there may be several disjoint subsets of processes such that processes in different subsets disagree on who are the current members of the group (but agreement is guaranteed within each subset).

According to a result in [CHT95], primary-partition group membership cannot be solved in asynchronous systems with failures, and this is because processes must agree on the content of the x^{th} view of a group. By allowing disagreement on the content of the x^{th} view, partitionable group membership services escape from this impossibility result, but they run into another problem: On one hand, their specification should be strong enough to preclude useless protocols, e.g., protocols that *capriciously* split processes into several concurrent views of the same group (and in particular into singleton sets); on the other hand, their specification should be weak enough to be implementable in asynchronous systems with failures. In the next two sections we show that the specifications of partitionable group membership service given in [DMS94, DMS95, EMS94, EMS95] do not satisfy the first requirement.

3.1 Capricious Splitting

The specification of the *Newtop* group membership service, given in [EMS94] and updated in [EMS95], is unsatisfactory because it can be satisfied by the following trivial protocol: Every process p initially installs $\{p\}$ as its first view of the group, and it never installs another view after this. This protocol simply splits all processes into singleton (one-member) views forever. Note that this total splitting occurs in *all* executions of this protocol, even those in which there are no process or communication failures (actual or suspected).

To preclude such trivial protocols, the formal specification of the partitionable group membership service given in Section 3 of [DMS94] includes the following "non-triviality" requirement (denoted M.3 in [DMS94]): For every subset of processes T and every process p, there is at least one execution in which p installs T as its view of the group. This requirement indeed prevents the trivial protocol above, but it allows group membership protocols that are equally useless, because all their executions, including those without any actual or suspected failures, result in a *permanent* total split. We sketch an example of such a protocol. Processes *a priori* agree on a total ordering $T_1, T_2, \cdots, T_{2^n}$ of all the subsets of processes. In the protocol, each process p first successively installs T_1, T_2, etc. as its group view, and then p immediately installs $\{p\}$ as its final view. Even though *all* the executions of this protocol result in a total and permanent split into singleton sets, the protocol satisfies the specification of partitionable group membership given in [DMS94].[3]

[3] In [DMS94, DMS95] the group membership protocol executes on top of a *causal communication module* which is responsible for multicasting messages, and there are some requirements on how these multicasts interact with view changes. To satisfy these requirements we assume that the group membership protocol above passes each multicast request (denoted $\mathcal{M}$-*multicast*) down to a causal communication

In response to this criticism, the non-triviality requirement of this specification was strengthened in [DMS95]. Roughly speaking, the revised requirement says that for every subset of processes T, every process $p \in T$, and every finite partial execution σ, there is an extension of σ in which p installs T as its group view. This stronger specification precludes the permanent split of processes into singleton sets but it still allows useless protocols. For example, it can be satisfied by the following protocol which capriciously splits processes into singleton views *for most of the time in all executions.* As before, processes *a priori* agree on a total ordering $T_1, T_2, \cdots, T_{2^n}$ of all the subsets of processes. Each process p successively installs T_1, T_2, etc. as its group view, skipping those subsets T such that $p \notin T$ ([DMS95] added the requirement that each process must be a member of every group view that it installs), and then p immediately installs $\{p\}$ as its view. After waiting for an arbitrarily long time, p repeats the above steps.[4] This protocol satisfies the stronger non-triviality requirement of [DMS95] but it is still useless: All its executions, including failure-free and suspicion-free ones, have an infinite number of intervals of arbitrary length each, during which processes are capriciously split into singleton sets.

3.2 Capricious View Changes

In general, the specification of a group membership protocol must regulate the installation of new views of a group. Firstly, it should require that if certain events occur (such as real or suspected failures, or requests to join or leave the group) then a new view of the group must be eventually installed to reflect these events. Secondly, it should also require that a new view of the group is installed only if certain events previously occurred: these are the events that justified the view change. This second type of requirement prevents the capricious installation of new views. Both types of requirements exist in the specification of the primary-partition group membership given in [RB91, Ric93, RB94]: They correspond to GMP-4 (the *liveness* requirement), and GMP-1 (the *validity* requirement), respectively.

In contrast, the specification of partitionable group membership given in [DMS94, DMS95] does *not* include the second type of requirement: It does not prevent the capricious installation of new group views. More precisely, the specification allows an arbitrary set of (operational) processes T to install T as a new group view, and to do so at any time and without any reason. This allows the arbitrary removal of processes that have not failed and, worse, that are not even suspected of having failed! This problem is recognized in [DMS95]: "Our membership framework allows capricious installations of views.... it does not restrict the removal of correct (and unsuspected) processes from the view."

---

module that satisfies the communication requirements given in Section 2.1 of [DMS94].

[4] We assume each $\mathcal{M}$-multicast is passed to the underlying causal communication module that satisfies the communication requirements given in Section 4 of [DMS95].

The above behavior is clearly undesirable. This flaw in the specification is partly due to the following shortcoming of the formal model: *Failure suspicions are generated but then completely ignored*, as we now explain. In [DMS94, DMS95] the group membership module executes on top of a causal communication module which is responsible for diffusing messages. When this communication module encounters some difficulty in delivering a message to some process q, it suspects that q is "disconnected" (i.e., it generates event $\mathcal{C}$-*suspect*(q)) and notifies the group membership module of this suspicion. Such notifications, however, are ignored in the specification of the group membership: None of the group membership requirements (denoted M.1 to M.6 in [DMS95]) refers to $\mathcal{C}$-*suspect* events. So, failure suspicion events have no specified effect on group membership. In order to integrate failure suspicions into the specification of group membership, the authors of [DMS95] state their intention to "analyze the framework in environments extended with failure detectors (e.g. the failure detectors discussed in [CT91])."

4 Discussion

The specification of any group membership service (and indeed of any service) should meet two requirements: It should be strong enough to guarantee that *any* protocol that satisfies it is indeed useful, and it should be weak enough to be solvable in a system with failures. The specifications given in [RB91, Ric93, RB94, DMS94, DMS95, EMS95] fail to meet one of the two requirements.

The specification of primary-partition group membership given in [RB91, Ric93, RB94] requires processes (1) to agree on the sequence of group views, and (2) to eventually install a new group view whenever a failure suspicion occurs. This is strong enough to be useful, but it is doubtful that it can be implemented in asynchronous systems [CHT95]: This is probably why there is a gap between what this specification requires and what is actually achieved by the "matching" group membership protocol given in [RB91, Ric93, RB94], a protocol that can cause collective suicide. On the other hand, the specifications of partitionable group membership given in [DMS94, DMS95] and in [EMS95] may be weak enough to be implementable, but they are not strong enough to prevent useless protocols that capriciously install views.

As a final remark, we would like to point out a disturbing trend. In our study of the group membership problem we noticed the following recurring theme: A group membership *protocol* would remain stable, while the *specification* of that protocol undergoes significant changes in content across successive versions. This is *prima facie* evidence that many of these group membership protocols (and sometimes their proofs of correctness) were written before their precise specification was actually determined and fixed. We believe that this practice is contrary to the principles of good system development.

Acknowledgement

We are grateful to Paul Ezhilchelvan, Dalia Malki, and Aleta Ricciardi for many useful discussions on [EMS94], [DMS94], and [RB94]. We also thank Vassos Hadzilacos for his helpful comments on an earlier draft of this paper.

Appendix

We explain why one cannot assign an arbitrary boolean value, *true* or *false*, to $\textsc{in-local}_p^x(q)$ at cuts c where $\mathsf{LocalView}_p^x$ is not defined. To do so, we first recall some definitions given in [RB94].

A process p can execute events that *add* or *remove* processes to the set of processes that p considers to be currently in the group. Each such event changes p's local view of the group, and $\mathsf{LocalView}_p^x$ is defined to be the x^{th} distinct version of p's local view. More precisely, $\mathsf{LocalView}_p^x$ is the set of processes in the group according to p after p executed a sequence of exactly x add or remove events. The paper states that, for any process q, $\textsc{in-local}_p^x(q)$ holds when $q \in \mathsf{LocalView}_p^x$, and that $\textsc{notdef'd}(\mathsf{LocalView}_p^x)$ holds if p has not yet defined its x^{th} local view.

Now take a consistent cut c such that p has done exactly $x-1$ view changes and then crashes. Since p never defines its x^{th} local view, $\textsc{notdef'd}(\mathsf{LocalView}_p^x)$ always holds. Take a process $q \in \mathsf{LocalView}_p^{x-1}$. By definition, $\textsc{in-local}_p^{x-1}(q)$ is *true* at c and at all the extensions of c. But what is the value of $\textsc{in-local}_p^x(q)$? The definition of $\textsc{in-local}_p^x(q)$ in [RB94] does not say. Moreover, assigning a fixed boolean value to $\textsc{in-local}_p^x(q)$ is problematic as we now show. Property GMP-3 implies the following two requirements:

$$c \models \Diamond \left(\textsc{in-local}_p^x(q) \Rightarrow \textsc{down}_q \vee (\mathsf{LocalView}_q = \mathsf{LocalView}_p^x)\right) \tag{1}$$

$$c \models \Diamond \left(\left(\neg\textsc{in-local}_p^x(q) \wedge \bigvee_{y<x} \textsc{in-local}_p^y(q)\right) \Rightarrow \Box\textsc{notdef'd}(\mathsf{LocalView}_q^x)\right) \tag{2}$$

Suppose we set $\textsc{in-local}_p^x(q)$ to *true*. In this case, the first formula reduces to:

$$c \models \Diamond \left(\textsc{down}_q \vee (\mathsf{LocalView}_q = \mathsf{LocalView}_p^x)\right)$$

This means that inevitably either q crashes or q installs a local view equal to $\mathsf{LocalView}_p^x$, a set which is not defined. This requirement is nonsensical.

Now suppose we set $\textsc{in-local}_p^x(q)$ to *false*. Since $\neg\textsc{in-local}_p^x(q)$ and $\textsc{in-local}_p^{x-1}(q)$ are *true* at c and at all the extensions of c, formula (2) implies $\Box\textsc{notdef'd}(\mathsf{LocalView}_q^x)$. This means that q can never install its x^{th} group view. This is also nonsensical.

References

[ADKM92a] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Membership algorithms for multicast communication groups. In *Proceedings of the 6th*

International Workshop on Distributed Algorithms (WDAG - 6), (LCNS, 647), pages 292–312, November 1992.

[ADKM92b] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Transis: a communication sub-system for high availability. In *Proceedings of the 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 76–84, Boston, July 1992.

[BDGB94] Özalp Babaoğlu, Renzo Davoli, Luigi-Alberto Giachini, and Mary Gray Baker. *RELACS: a communications infrastructure for constructing reliable applications in large-scale distributed systems.* BROADCAST Project deliverable report, 1994. Department of Computing Science, University of Newcastle upon Tyne, UK.

[Bir93] Kenneth P. Birman. The process group approach to reliable distributed computing. *Communication of the ACM*, 9(12):36–53, December 1993.

[CHT95] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. Impossibility of group membership in asynchronous systems. Technical Report 95-1533, Computer Science Department, Cornell University, Ithaca, New York 14853, August 1995.

[CL85] K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985.

[Cri87] Flaviu Cristian. Issues in the design of highly available computing services. In *Annual Symposium of the Canadian Information Processing Society*, pages 9–16, July 1987. Also IBM Research Report RJ5856, July 1987.

[Cri91] Flaviu Cristian. Reaching agreement on processor group membership in synchronous distributed systems. *Distributed Computing*, 4(4):175–187, April 1991.

[CT91] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for asynchronous distributed systems. In *Proceedings of the 10th ACM Symposium on Principles of Distributed Computing*, pages 325–340, August 1991. Available by anonymous ftp from `ftp.cs.cornell.edu` in `pub/chandra/failure.detectors.algorithms.dvi.Z`. To appear in the *Journal of the ACM.*

[DMS94] Danny Dolev, Dalia Malki, and Ray Strong. An asynchronous membership protocol that tolerates partitions. Technical Report CS94-6, Institute of Computer Science, The Hebrew University of Jerusalem, 1994.

[DMS95] Danny Dolev, Dalia Malki, and Ray Strong. A framework for partitionable membership service. Technical Report CS95-4, Institute of Computer Science, The Hebrew University of Jerusalem, 1995.

[EMS94] Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: a fault-tolerant group communication protocol. Technical report, Department of Computing Science, University of Newcastle upon Tyne,UK, July 1994.

[EMS95] Paul D. Ezhilchelvan, Raimundo A. Macêdo, and Santosh K. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, Vancouver, BC, Canada, June 1995.

[HS95] Matti A. Hiltunen and Richard D. Schlichting. Properties of membership services. In *Proceedings of the 2nd International Symposium on Autonomous Decentralized Systems*, Phoenix, AZ, April 1995.

[JFR93] Farnam Jahanian, Sameh Fakhouri, and Ragunathan Rajkumar. Processor group membership protocols: specification, design and implementation. In *Proceeding of the 12th IEEE Symposium on Reliable Distributed Systems*, pages 2–11, Princeton, October 1993.

[KT91] M. Frans Kaashoek and Andrew S. Tanenbaum. Group communication in the amoeba distributed operating system. In *Proceedings of the 11th International Conference on Distributed Computer Systems*, pages 222–230, Arlington, TX, May 1991.

[Mat88] Friedemann Mattern. Virtual time and global states of distributed systems. In Michel Cosnard, Patrice Quinton, Yves Robert, and Michel Raynal, editors, *Proceedings of the International Workshop on Parallel and Distributed Algorithms*, pages 215–226. North-Holland, October 1988.

[MPS91] Shivakant Mishra, Larry L. Peterson, and Richard D. Schlichting. A membership protocol based on partial order. In *Proceedings of the IEEE International Working Conference on Dependable Computing For Critical Applications*, pages 137–145, Tucson, AZ, February 1991.

[MSMA94] P.M. Melliar-Smith, Louise Moser, and Vivek Agrawala. Processor membership in asynchronous distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 5(5):459–473, May 1994.

[RB91] Aleta Ricciardi and Kenneth P. Birman. Using process groups to implement failure detection in asynchronous environments. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages

341–352, 1991. Also available as technical report 93-1328, Department of Computer Science, Cornell University.

[RB94] Aleta Ricciardi and Kenneth P. Birman. Process membership in asynchronous environments. Available by anonymous ftp from `ftp.cs.cornell.edu` in `pub/aleta/AsyncMembService.ps`, April 1994.

[Ric93] Aleta Ricciardi. *The group membership problem in asynchronous systems.* PhD thesis, Department of Computer Science, Cornell University, USA, January 1993.

[vRBC+93] Robbert van Renesse, Kenneth P. Birman, Robert Cooper, Bradford Glade, and Patrick Stephenson. The horus system. In Kenneth P. Birman and Robbert van Renesse, editors, *Reliable Distributed Computing with the Isis Toolkit*, pages 133–147. IEEE Computer Society Press, Los Alamitos, CA, 1993.