

# No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement in Message-Passing Systems

Sylvain Gay  

École Normale Supérieure, Paris, France

Achour Mostéfaoui  

LS2N, Nantes Université, France

Matthieu Perrin  

LS2N, Nantes Université, France

---

## Abstract

This paper explores the relationship between broadcast abstractions and the  $k$ -set agreement ( $k$ -SA) problem in crash-prone asynchronous distributed systems. It specifically investigates whether any broadcast abstraction is computationally equivalent to  $k$ -SA in message-passing systems.

A key contribution of the paper is the delineation of the realm of “meaningful” broadcast abstractions, through the introduction of two new symmetry properties: *compositionality* and *content-neutrality*, inspired by the principle of network neutrality. Such preciseness in definition is essential for this paper’s scope, as our aim is not to characterize the computing power of a specific broadcast abstraction, but rather to explore the domain of broadcast abstractions as a whole, in search of a broadcast abstraction with certain characteristics. The paper’s main contribution is the proof that no broadcast abstraction, which is both content-neutral and compositional, is computationally equivalent to  $k$ -set agreement when  $1 < k < n$ , in the crash-prone asynchronous message-passing model. To the best of our knowledge, this result represents the first instance of showing that a coordination problem cannot be expressed by an equivalent broadcast abstraction. It does not establish the absence of an implementation, but rather the absence of a specification that possesses certain properties.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models; Computer systems organization → Dependable and fault-tolerant systems and networks; Networks → Network properties

**Keywords and phrases** Agreement problem, Asynchronous system, Broadcast abstraction, Communication abstraction, Compositionality, Message-passing system, Network neutrality, Process crash,  $k$ -Set agreement, Wait-free model, Total order broadcast

**Digital Object Identifier** 10.4230/LIPIcs.OPODIS.2024.21

**Funding** This work was partially supported by the French ANR project ByBloS (ANR-20-CE25-0002-01) and the CominLabs Labex project PriCLeSS (ANR-10-LABX-07-81).

## 1 Introduction

### 1.1 From Send/Receive to Communication Abstractions

This paper considers distributed systems consisting of a set of asynchronous processes prone to crash failures. These processes communicate by sending and receiving messages across an asynchronous network and must cooperate to achieve a common goal. What makes distributed computing challenging is that the dynamics of the underlying network on which the distributed application operates are beyond the programmer’s direct control. This necessitates treating the environment as a “hidden input” forcing processes to manage uncertainty at runtime [20]. To facilitate the design of advanced algorithms in this unpredictable setting, it is usual to



© Sylvain Gay, Achour Mostéfaoui, and Matthieu Perrin;  
licensed under Creative Commons License CC-BY 4.0

28th International Conference on Principles of Distributed Systems (OPODIS 2024).

Editors: Silvia Bonomi, Letterio Galletta, Etienne Rivière, and Valerio Schiavoni; Article No. 21; pp. 21:1–21:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

define appropriate communication abstractions, that allow modularity and help mitigate uncertainty by restricting communication patterns that may occur at a higher abstraction level.

In crash-prone asynchronous distributed systems, a significant source of uncertainty stems from the divergent perceptions of the event set (i.e., send and receive events) among different processes. *Broadcast abstractions*, which enable processes to transmit a message to all participants within the same operation, alleviate this issue by ensuring consistent and reliable communication across different nodes, thereby simplifying the complexity of managing individual send/receive operations and enhance fault tolerance by reducing the impact of node failures. Hence, message broadcasts (at least by correct processes) constitute a set of global events for which all correct processes eventually agree they took place, thereby establishing a system-wide communication foundation that enables cooperation and synchronization.

Another source of uncertainty arises from the disparate order in which different participants may receive messages, leading to varied perceptions of the global state of the system. Several communication abstractions have been defined by enforcing properties on the message delivery order. FIFO and Causal Ordering are examples of such properties at the heart of FIFO-broadcast and Causal-broadcast [3, 21]. These abstractions facilitate the construction of distributed objects, like causal memory in asynchronous message-passing systems [2].

**A remark on vocabulary.** Throughout this paper, to avoid confusion, we distinguish between the terms “send” and “receive”, which denote low-level point-to-point communication primitives applied to individual messages, and “broadcast” and “deliver”, which describe the higher-level operations of one-to-all broadcast abstractions. Consequently, the terms “receive” and “deliver” are not used interchangeably in the context of this paper.

## 1.2 Characterizing Coordination Problems with Broadcast Abstractions

This paper follows the quest of identifying broadcast abstractions that characterize the major fundamental problems in distributed computing. Specifically, we aim to determine broadcast abstractions that are computationally equivalent to particular synchronization problems in a crash-prone asynchronous message-passing system. This equivalence means that a deterministic algorithm based on the broadcast abstraction can resolve the synchronization problem regardless the number of crash failures, and vice versa.

A well-known such characterization is the equivalence between *Total Order Broadcast* and the *consensus* problem [6]. Consensus is a fundamental problem of distributed computing, that provides processes with a single operation, denoted `propose()`. This operation allows each process to propose a value and decide on a value. The defining properties of this problem are as follows: if a process invokes `propose(v)` and does not crash, it will decide on a value (termination); no two processes will decide on different values (agreement); and the decided value must have been proposed by a process (validity). One of the primary practical applications of consensus is to maintain consistency across replicated machines in a message-passing system. However, State Machine Replication (SMR) [23] typically builds on an intermediate communication abstraction, the well-known and powerful *Total Order Broadcast* abstraction [18]. This abstraction ensures that the order of message delivery is consistent across all processes.

The consensus problem is famously unsolvable in an asynchronous distributed system, even under the assumption that at most one process may crash [10]. The same holds for Total Order Broadcast. Indeed, both abstractions are computationally equivalent: on the one hand, Total Order Broadcast allows an easy implementation of a consensus object; on

the other hand, it can be implemented using consensus objects. In a sense, Total Order Broadcast precisely “characterizes” the essence of the consensus problem. In a similar vein, *Mutual Broadcast* was recently proposed as a broadcast abstraction equivalent to read/write atomic registers [8]. Moreover, *Pair Broadcast* characterizes the computational power of both test-and-set and consensus between two processes [9]. Such characterizing broadcast abstractions are instrumental for understanding the fundamentals of distributed computing problems by reducing their complexity into a logical property about the order in which different processes perceive the global events (a.k.a. broadcast invocations) occurring in the system.

### 1.3 On the $k$ -Set Agreement Side

Specifically, this paper delves into characterizing the  $k$ -set agreement problem ( $k$ -SA), a generalization of consensus introduced by S. Chaudhuri in [7]. In  $k$ -SA, the agreement property is weakened as follows: processes are allowed to collectively decide up to  $k$  different values. Here,  $k$  represents the maximum disagreement in the number of different values that can be decided. The smallest value  $k = 1$  corresponds to consensus. As  $k$  increases, the problem becomes less constrained and may become easier to solve. However, it still embodies numerous complexities and challenges of distributed systems. It remains insoluble in a crash-prone asynchronous system when  $k \leq t$ , where  $t$  is the maximum number of processes in the system that may crash [4, 12, 22].

The exploration of a broadcast abstraction that characterizes  $k$ -SA was initiated in a work dedicated to the shared-memory model, which proposed *k-Bounded Order Broadcast* ( $k$ -BO Broadcast in short) [13]. The  $k$ -BO Broadcast abstraction limits the disagreement on the message delivery order among processes. Specifically, its ordering property asserts that every set of  $k + 1$  messages contains two messages delivered in the same order by all processes. In the special case where  $k = 1$ , it boils down to Total Order Broadcast.

In crash-prone asynchronous systems where processes additionally have access to a shared memory composed of atomic read/write registers,  $k$ -BO Broadcast is computationally equivalent to  $k$ -set agreement. However, this equivalence in shared memory does not inherently translate to message-passing systems. Indeed, although  $k$ -BO broadcast can be used to solve  $k$ -set agreement on its own, it remains unproven whether it can be implemented using solely  $k$ -set agreement objects and send/receive operations. While consensus is strong enough to emulate atomic registers in a message-passing system,  $k$ -set agreement, for  $k > 1$ , is unable to emulate shared memory. Indeed, it has been proved that on one hand,  $k$ -SA and a problem called the  $k$ -simultaneous-agreement are equivalent in shared memory systems [1], and on the other hand, the  $k$ -simultaneous-agreement problem is harder than  $k$ -SA in message-passing systems where a shared memory emulation is not possible [5]. A corollary of this paper is that the implementation of  $k$ -BO broadcast on top of  $k$ -SA is not feasible in message-passing systems.

**Problem Statement.** Consequently, the existence of a broadcast abstraction that precisely characterizes  $k$ -set agreement in message-passing systems remains uncertain. This paper investigates the following question: *Does there exist a broadcast abstraction computationally equivalent to  $k$ -set agreement in crash-prone asynchronous message-passing systems?*

## 1.4 Contributions

**Introducing symmetry properties.** A simplistic approach to the discussed question might propose the following ordering property: “At most  $k$  distinct messages can be delivered as the first messages by the processes.” Indeed, on the one hand, a  $k$ -SA object can select the set of messages eligible for initial delivery; and on the other hand,  $k$ -SA can be trivially solved by broadcasting all proposed values and deciding on the first delivered ones, hence establishing equivalence.

However, such a solution is deemed “unsatisfactory”, as an instance of this broadcast abstraction would effectively solve  $k$ -SA once, before the ordering property becomes meaningless. Hence, an iterative resolution of  $k$ -SA would necessitate a distinct broadcast instance for each  $k$ -SA object implemented. This requirement deviates from the traditional model of interaction with message-passing systems, where a communication abstraction serves as a system-wide service shared among multiple algorithms for solving higher-level tasks. In contrast, the proposed solution discriminates among messages, specifically restricting the specified ordering property to only the initial messages. This restriction violates the principle of “network neutrality,” which advocate for equal treatment of all messages [24]. By doing so, this limitation undermines the ability to construct modular higher-level systems composed of independent and composable components.

Hence, before delving into our main problem statement, it is necessary to properly define the research space by clarifying another important question: *What constitutes a broadcast abstraction?* A major contribution of this article is the introduction of two symmetry properties drawing inspiration from the principle of network neutrality: *compositionality* and *content-neutrality*. Compositionality ensures that a broadcast abstraction does not discriminate based on system-wide knowledge that would be inaccessible to individual components built upon it. This property effectively prevents the scenario described in our previous example and allows for the construction of composable applications. Content-neutrality ensures that the behavior of a broadcast abstraction does not depend on the content of the messages.

It is important to note that the principle of network neutrality prohibits discrimination based on other aspects, such as the identity of message emitters or recipients. We will not explore these aspects in this paper because they are not pertinent to our argument concerning  $k$ -set agreement. We leave as future work the investigation of whether such properties impact the characterization of other synchronization problems within the framework of broadcast abstractions.

**A Non-existence Result.** Having defined what constitutes an appropriate broadcast abstraction, we are now equipped to address our problem statement, to which we provide a negative answer: we demonstrate that no broadcast abstraction, which is both content-neutral and compositional, is computationally equivalent to  $k$ -SA for  $1 < k < n$  in a crash-prone asynchronous message-passing system. To the best of our knowledge, this research presents the first instance where a coordination problem has been proven to lack an equivalent broadcast abstraction.

**Paper Organization.** The remainder of this paper is organized as follows. Section 2 delineates the crash-prone asynchronous message-passing distributed computing model pertinent to our results. Subsequently, Section 3 defines permissible broadcast abstractions, introducing the novel symmetry properties. Section 4 then establishes that no content-neutral and compositional broadcast abstraction is computationally equivalent to  $k$ -set agreement for  $1 < k < n$ . Finally, Section 5 concludes the paper.

## 2 The Asynchronous Crash-Prone Message-Passing Model

The computing model is the classical asynchronous crash-prone message-passing model.

**Process Model.** The computing model consists of a set  $\Pi$  of  $n$  sequential processes denoted  $p_1, \dots, p_n$ . Each process operates asynchronously, meaning it progresses at its own speed, which is arbitrary, unknown to other processes, and may vary through time. A process may halt prematurely (crash failure) but executes its local algorithm correctly until it possibly crashes. We do not assume any bound on the number of processes that may crash, hence  $t = n - 1$ . A process that crashes in a run is said to be *faulty*. Conversely, a process is called *correct* or *non-faulty* if it does not crash.

**Communication Model.** Communication between each pair of processes occurs through two uni-directional channels, one for each direction. Consequently, the network is complete: any process  $p_s$  can directly send a message to any process  $p_r$  (including itself). Each channel is reliable (free from loss, corruption, or message creation), not necessarily FIFO (First-In/First-Out), and asynchronous (messages have finite but unbounded transit times). It is important to note that, due to asynchrony in processes and message delivery, no process can ascertain whether another process has crashed or is merely slow. A process  $p_s$  invokes the operation “send  $m$  to  $p_r$ ” to send a message whose content is  $m$  to  $p_r$ . The event “receive  $m$  from  $p_s$ ” occurs at  $p_r$  upon receiving a message whose content is  $m$  from  $p_s$ . Although messages may share content, each sent message is unique. By a slight abuse of language, when the distinction between a message and its content is immaterial, we say that “a process  $p_i$  sends (resp. receives) a message  $m$ ” when  $p_i$  sends or receive a message whose content is  $m$ . The communication channels are governed by the following properties:

**SR-Validity.** If the event “receive  $m$  from  $p_s$ ” occurs at  $p_r$ , then  $p_s$  has indeed invoked the operation “send  $m$  to  $p_r$ ”.

**SR-No-Duplication.** No process receives the same message more than once.

**SR-Termination.** If a process  $p_s$  sends a message  $m$  to a correct process  $p_r$ , then  $p_r$  will eventually receive  $m$  from  $p_s$ .

**Notation.** The acronym  $\mathcal{CAMP}_n[\emptyset]$  denotes the described Crash-prone Asynchronous Message-Passing model without additional computational power.  $\mathcal{CAMP}_n[H]$  represents  $\mathcal{CAMP}_n[\emptyset]$  enhanced with the additional computational power denoted by  $H$ . For instance,  $\mathcal{CAMP}_n[k\text{-SA}]$  denotes the model  $\mathcal{CAMP}_n[\emptyset]$  in which processes have access to as many instances of the  $k$ -set agreement object as needed. Similarly, if  $B$  represents a broadcast abstraction, then  $\mathcal{CAMP}_n[B]$  refers to the  $\mathcal{CAMP}_n[\emptyset]$  model in which processes can broadcast and deliver messages via the abstraction  $B$ .

**Execution.** An execution  $\alpha$  is a sequence of steps, each represented as a pair  $\langle p_i : a \rangle$ , where  $p_i \in \Pi$  represents a process, and  $a$  is an action occurring at  $p_i$ . These actions can be local computations, the invocation of primitives (such as send), the triggering of local events (including message receptions), as well as invocations and responses of high-level operations as specified in the enriching hypothesis  $H$ . Examples of such high-level operations include proposing or deciding on a value in a  $k$ -SA object.

We define an execution  $\alpha$  as being admitted by the model  $\mathcal{CAMP}_n[H]$  if it satisfies several criteria: it must adhere to the three properties of the communication channels, namely SR-VALIDITY, SR-NO-DUPLICATION, and SR-TERMINATION; it must conform to all

## 21:6 No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement

properties specified by  $H$  and the high-level abstractions it provides (for example, if  $k$ -set agreement objects are available, they should verify the  $k$ -SA-VALIDITY,  $k$ -SA-AGREEMENT and  $k$ -SA-TERMINATION properties defined in Section 4.1); and it must be *well-formed* with respect to the algorithm it executes, as delineated by the following definition.

► **Definition 1 (Well-Formed Executions).** Consider  $\mathcal{A}$ , a deterministic algorithm that implements a high-level abstraction  $A$  within the  $\mathcal{CAMP}_n[H]$  model. An execution is deemed *WELL-FORMED* with respect to  $\mathcal{A}$  if it fulfills the following conditions:

- Only processes labeled from  $p_1$  to  $p_n$  take actions in  $\alpha$ ;
- A process never invokes an operation of  $A$  before returning from its previous invocations;
- The actions undertaken by any process between the invocation of an operation on  $A$  and its corresponding response (if one exists) must align with the actions specified by  $\mathcal{A}$ .

Given a high-level abstraction  $X$  (for example a broadcast abstraction), we say that an execution  $\alpha$  is admitted by  $X$  if it is admitted by the model  $\mathcal{CAMP}_n[X]$ .

### 3 Defining Admissible Broadcast Abstractions

#### 3.1 Interface of broadcast abstractions

A broadcast abstraction denoted as  $B$ , enables correct processes to broadcast messages that are guaranteed to be delivered at least to all correct processes. Consequently, all broadcast abstractions share the same interface, comprising a single operation named `broadcast` and an event called `deliver`.

A process  $p_i$  invokes the operation “ $B.\text{broadcast}(m)$ ” to utilize  $B$  for broadcasting a message whose content is  $m$ . This is referred to as  $p_i$   $B$ -broadcasting a message whose content is  $m$ . Subsequently, the event “ $B.\text{deliver } m \text{ from } p_i$ ” might be triggered at some processes  $p_j$ , leading us to say that  $p_j$   $B$ -delivers a message from  $p_i$ , whose content is  $m$ . Analogous to the send/receive interface, it is assumed that each broadcast message is unique, regardless of having identical content. However, for the seek of readability, we amalgamate a message and its content whenever the distinction is immaterial. The set of all messages that can be broadcast during an execution is denoted by  $\mathbb{M}$ . The following properties must be verified by all broadcast abstractions.

**BC-Validity.** If a process  $p_i$   $B$ -delivers  $m$  from  $p_j$ , then it is guaranteed that  $p_j$  has previously  $B$ -broadcast  $m$ .

**BC-No-Duplication.** A process will not  $B$ -deliver the same message more than once.

**BC-Local-Termination.** If a correct process invokes  $B.\text{broadcast}(m)$ , it will eventually return from this invocation.

**BC-Global-CS-Termination.** If a correct process  $B$ -broadcasts  $m$ , then all correct processes will eventually  $B$ -deliver  $m$ .

The first two properties mentioned are classical safety properties and share the same definitions as their send/receive counterparts. The third property is a classical liveness property. It is important to note that the BC-GLOBAL-CS-TERMINATION property only applies to correct processes. (The abbreviation “CS”, standing for *correct sender*, emphasizes that this property is contingent on the sender’s correctness.) Consequently, if a process  $p_i$  crashes during its execution of `broadcast`( $m$ ), it is permissible for some processes to deliver  $m$  while others do not, unless otherwise specified. This specification choice is intentionally made to allow for flexible definitions of liveness properties in broadcast abstractions.

In particular, the most basic broadcast abstraction that can be defined, only verifies the four properties defined above. In the  $\mathcal{CAMP}_n[\emptyset]$  model, its implementation involves simply sending messages to all participants. For this reason, it is commonly referred to as Send-To-All Broadcast.

**Remark on Expressiveness.** Set-Constrained-Delivery Broadcast (SCD Broadcast) [14] and its extension  $k$ -SCD Broadcast [13] are two examples of broadcast abstractions whose specification slightly deviate from the propose interface. Indeed, these abstractions deliver messages not individually, but within unordered sets of messages, hence the name “SCD”. While it is easy to generalize the definitions and the proofs to accommodate this particularity, we have chosen not to pursue this generalization for the sake of clarity and simplicity.

**A local ordering property.** When considered together, the BC-VALIDITY and BC-GLOBAL-CS-TERMINATION properties ensure that a step  $\langle p_i : B.\text{broadcast}(m) \rangle$  executed by a correct process  $p_i$  is always followed by a step  $\langle p_i : B.\text{deliver } m \text{ from } p_i \rangle$ . In a similar vein, the BC-LOCAL-TERMINATION property guarantees that the  $B$ -broadcasting step is consistently succeeded by  $\langle p_i : \text{return from } B.\text{broadcast}(m) \rangle$ . However, there is no inherent order between the delivery of its own message  $m$  by  $p_i$ , and  $p_i$  returning from its  $B.\text{broadcast}$  invocation. Once again, this specification choice is deliberately made to accommodate flexible definitions of broadcast abstractions. For instance, certain abstractions may require that  $B.\text{broadcast}$  returns immediately, or they may wait until the broadcast message has been delivered, while others may delegate the decision to the implementation. Nevertheless, it is occasionally beneficial to reason based on a fixed total order among the three events. Adopting the terminology suggested in [8], we augment all broadcast abstractions with a mix-in  $B.\text{sync-broadcast}(m)$ , defined as:  $B.\text{broadcast}(m); \text{wait}(m \text{ has been } B\text{-delivered locally})$ . For every message  $m$   $B$ -broadcast by each correct process  $p_i$ , the following three steps occur sequentially:  $\langle p_i : B.\text{sync-broadcast}(m) \rangle$ , followed by  $\langle p_i : B.\text{deliver } m \text{ from } p_i \rangle$ , and then  $\langle p_i : \text{return from } B.\text{sync-broadcast}(m) \rangle$ .

### 3.2 Symmetry Properties of Broadcast Abstractions

Broadcast abstractions can be characterized by additional predicates on the set of executions they admit. Typically, these predicates fall into two categories. On one hand, liveness predicates ensure message delivery in scenarios not covered by Send-To-All Broadcast. Examples of this include the definitions of Reliable Broadcast and Uniform Reliable Broadcast [11]. On the other hand, safety predicates concern the relative order in which processes deliver messages. Examples in this category are FIFO Broadcast, Causal Broadcast [3, 21], Mutual Broadcast [8], Pair Broadcast [9],  $k$ -Bounded Order Broadcast [13], and Total Order Broadcast [18].

As highlighted in the Introduction, not all predicates are equally appropriate for the design of a broadcast abstraction. In this section, we introduce two novel symmetry properties inspired by the broader principle of “network neutrality”. Network neutrality advocates, among other tenets, that network services should not discriminate based on the content, sender, or usage of the messages they transmit. While concerns regarding network neutrality often arise in discussions about non-functional aspects of message routing, they hold significant relevance for the functional design of broadcast abstractions. Within this framework, we interpret network neutrality to include two essential symmetry properties: *Compositionality* and *Content Neutrality*. These properties assert that the broadcast abstraction should impartially treat all messages, irrespective of their usage or content.

**Compositionality.** Building upon the broadcast described in the Introduction, one might propose characterizing iterated  $k$ -SA using an abstraction  $k$ -Stepped Broadcast, characterized by the following ordering property: “for each  $a$ , define  $S_a$  as the set containing the  $a^{\text{th}}$  message broadcast by each process; then there are at most  $k$  messages  $m \in S_a$  such that some process delivers  $m$  before any other message in  $S_a$ ”. Now, the ordering of messages within each  $S_a$  set could determine the set of values decided on a sequence of  $k$ -SA objects, and conversely, thereby establishing equivalence.

However, since the ordering property only governs specific sets of messages, it imposes an overly precise communication pattern (lock-step pattern), restricting its utility for constructing modular higher-level systems. Consider, for instance, a system composed of two modules built upon the same service that provides this broadcast abstraction: the iterated  $k$ -SA algorithm described above and a messaging service utilizing only the Send-To-All capabilities of  $k$ -Stepped Broadcast. Each application employs only a distinct subset of the system’s messages, so the messages used by the messaging service interfere with the communication pattern followed by the iterated  $k$ -SA algorithm. Unless a global counter, shared between the two applications, is used to track the number  $a$  of broadcast messages, the modules cannot fully benefit from the offered ordering property. This limitation hinders their independent design and composition.

Compositionality is the property required for the implementation of composable algorithms or applications on top of a broadcast abstraction. Each higher-level construction uses only a subset of the messages broadcast at the lower level. Compositionality ensures that each of these message sets maintains the same ordering properties as those of the entire message set. This is achieved by requiring that the restriction of an admissible execution to any subset of its messages remains an admissible execution.

► **Definition 2 (Compositionality).** *A broadcast abstraction  $B$  is compositional if, for all executions  $\alpha$  admissible by  $B$ , and for any set of messages  $M$ , the restriction of  $\alpha$  onto the messages of  $M$  is also admissible by  $B$ .*

To exemplify the compositionality property, let us demonstrate that  $k$ -BO Broadcast is compositional. Indeed, its ordering property is defined by a predicate  $P$  that must be satisfied by any set  $S$  of messages. Specifically,  $P(S)$  stipulates that if  $S$  contains at least  $k + 1$  messages, then at least two of these messages must be delivered in the same order by all processes. Consider an execution  $\alpha$  admissible by  $k$ -BO Broadcast, with its set of sent messages denoted as  $M_\alpha$ . For any subset  $M \subseteq M_\alpha$  of these messages, every subset  $S$  of  $M$  is also a subset of  $M_\alpha$ , ensuring  $P(S)$  is satisfied, which is the condition imposed by compositionality. This logical framework can be applied to all broadcast abstractions defined by a predicate on the relative order of broadcast and deliver events, independent of the context of the complete execution. Notably, this encompasses all broadcast abstractions mentioned in the Introduction and, to the best of our knowledge, all broadcast abstractions currently described in the literature.

Conversely, the limitations of compositionality can be highlighted by revisiting our initial counter-example involving  $k$ -Stepped Broadcast. Consider an execution  $\alpha$  where two processes,  $p_0$  and  $p_1$ , engage in the 1-Stepped-broadcasting of two messages each:  $m_i$  and  $m'_i$ . In  $\alpha$ ,  $p_0$  delivers the messages  $[m_0, m'_0, m_1, m'_1]$  in this order. Simultaneously,  $p_1$  delivers the sequence  $[m_0, m_1, m'_0, m'_1]$ . Although both processes deliver  $m_0$  before  $m_1$  and  $m'_0$  before  $m'_1$ , conforming to the 1-stepped predicate applied to the sets  $S_1 = \{m_0, m_1\}$  and  $S_2 = \{m'_0, m'_1\}$ , the execution’s restriction to the subset  $M = \{m'_0, m_1\}$  fails to maintain this order. This issue arises because the definition quantifies over the sequence number  $a$  of the broadcast messages, which is only contextually relevant within the full scope of the execution and varies when subsets of messages are considered.



**Content Neutrality.** The second property asserts that the defining predicates of a broadcast abstraction should be applicable based solely on the occurrence of broadcast and delivery events during an execution, independent of the message’s content. Hence, if some messages get substituted by other within an execution, it should not hinder the admissibility of the execution. Content neutrality then stipulates that an admissible execution must remain admissible even when some of its messages are replaced.

► **Definition 3 (Content-Neutrality).** *A broadcast abstraction  $B$  is content-neutral if, for all executions  $\alpha$  admissible by  $B$ , and all injective endofunctions  $r$  on the set of messages, the execution obtained by replacing all messages  $m$  by  $r(m)$  in  $\alpha$ , is also admissible by  $B$ .*

It is important to note that while all broadcast abstractions mentioned in the Introduction adhere to the content-neutrality property, this is not necessarily true for all broadcast abstractions found in the literature. For instance, Generic Broadcast [17] supposes that the messages it transmits encapsulate a *command*, i.e., an operation invocation on a replicated state machine implemented using the broadcast. In the vein of Generalized Paxos [16], processes only need to agree on a common delivery order for pairs of non-commuting commands, as executing commuting commands in different orders does not compromise the consistency of the implemented data structure. However, *specifying* such a broadcast necessitates differentiating between messages, which violates content neutrality.

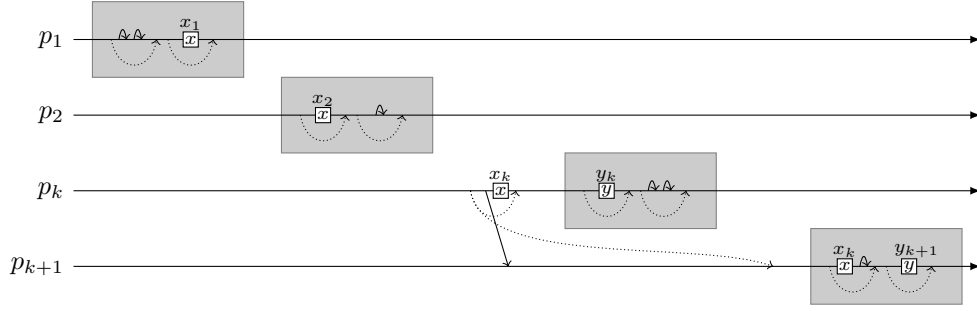
Returning to the present paper, it would be straightforward to propose a broadcast abstraction equivalent to  $k$ -set agreement that is not content-neutral. For example, one could enforce an ordering property that only applies to messages of a special type  $\text{SA}(ksa, v)$ , where  $ksa$  uniquely identifies a  $k$ -SA object and  $v$  is a value proposed to  $ksa$ . This would require that, for each  $ksa$ , at most  $k$  distinct messages of the form  $\text{SA}(ksa, \_)$  are delivered first by any process. In the following section, we focus exclusively on content-neutral broadcast abstractions.

## 4 On Characterizing $k$ -Set Agreement

Having defined what constitutes a symmetric broadcast abstraction, we are now equipped to address our problem statement, to which we provide a negative answer: we demonstrate that no broadcast abstraction, which is both content-neutral and compositional, is computationally equivalent to  $k$ -set agreement for  $1 < k < n$  in an asynchronous message-passing system where any number of processes may crash. It is clear that for  $k = 1$ , 1-SA boils down to consensus, which is characterized by Total Order Broadcast. Conversely, for  $k = n$ ,  $n$ -set agreement can be trivially solved without any communication, rendering it equivalent to Send-To-All Broadcast in a message-passing system. As previously mentioned in the Introduction, shared memory can be emulated when  $t < \frac{n}{2}$ , in which case  $k$ -SA is equivalent to  $k$ -BO Broadcast. Furthermore,  $k$ -SA can be solved for certain values of  $k$  and  $t$  (e.g. when  $t < n - k$ ), rendering it equivalent to Send-To-All Broadcast. This paper does not presuppose a maximum number of crash failures, implying that any process may crash (i.e  $t = n - 1$ ).

We begin by recalling the definition of  $k$ -set agreement in Section 4.1. The ensuing proof is structured as a *reductio ad absurdum*. We hypothesize the existence of a broadcast abstraction  $B$  satisfying the aforementioned conditions. Two deterministic reduction algorithms are then considered:  $\mathcal{A}$ , which implements  $k$ -set agreement in the model  $\mathcal{CAMP}_n[B]$ , and  $\mathcal{B}$ , which implements  $B$  in the model  $\mathcal{CAMP}_n[k\text{-SA}]$ . For any  $N \in \mathbb{N}$ , Section 4.2 constructs an execution  $\alpha_{k,N,B,\mathcal{B}}$  (as defined in Definition 4 and illustrated on Figure 1) of  $\mathcal{B}$ , wherein each process  $B$ -delivers  $N$  of its own messages before any messages from other processes.

## 21:10 No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement



■ **Figure 1** Illustration of the adversarial execution  $\alpha_{k,N,B,B}$  for  $k = 3$  and  $N = 2$ , extending up to Line 7 of Algorithm 1. Within the  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  model, plain arrows signify sent and received messages, while white squares denote propositions on  $k$ -SA objects, with their respective decided values indicated above. In the context of the  $\mathcal{CAMP}_{k+1}[B]$  model, simulated by Algorithm  $\mathcal{B}$ , dotted arrows represent  $B$ -broadcast and  $B$ -delivered messages. Notably, if the initial message  $B$ -broadcast by  $p_k$  is issued by an algorithm and the final  $N$  messages of each process are used by a different algorithm, then this latter algorithm is unable to resolve the  $k$ -set agreement.

Subsequently, in Section 4.3, we demonstrate that sufficiently large values of  $N$  inhibit  $\mathcal{A}$  from effectively resolving  $k$ -set agreement, thereby leading to a contradiction. Due to space constraints, the proofs of the lemmas are provided in the appendix.

### 4.1 Definition of $k$ -Set Agreement

$k$ -Set agreement, first introduced by S. Chaudhuri in [7] (refer to [19] for a comprehensive survey of  $k$ -set agreement in various contexts), was conceptualized to analyze the relationship between the maximum number of allowable process failures ( $t$ ) and the feasible degree of agreement ( $k$ ) among processes. Here, a lower value of  $k$  signifies a higher degree of agreement, with the ultimate agreement being  $k = 1$ , which corresponds to consensus.

The  $k$ -Set agreement problem is a one-shot agreement problem that equips processes with a singular operation, denoted `propose()`. When a process  $p_i$  invokes `ksa.propose( $v_i$ )` on a  $k$ -SA object  $ksa$ , it is said to “propose the value  $v_i$  to  $ksa$ ”. This operation yields a return value  $v$ , at which point the invoking process is described as “deciding  $v$  on  $ksa$ ”, and “ $v$  becomes a decided value”. In other words, the steps  $\langle p_i : \text{return } w \text{ from } ksa.\text{propose}(v) \rangle$  and  $\langle p_i : ksa.\text{decide}(w) \rangle$  are interpreted as synonymous. It is a standard assumption that each process is limited to a single invocation of `propose()` on any given  $k$ -SA object, ensuring the problem’s one-shot nature.  $k$ -Set agreement is defined by the following properties.

**$k$ -SA-Validity.** If a process decides a value  $v$ , then  $v$  was proposed by some process.

**$k$ -SA-Agreement.** No more than  $k$  distinct values are decided upon by the processes.

**$k$ -SA-Termination.** Every non-faulty process that invokes `propose()` eventually decides.

### 4.2 Definition of the adversarial scheduler

For brevity in this subsection, we pose  $k > 1$  and  $N > 0$ . Additionally, we postulate the existence of a deterministic algorithm  $\mathcal{B}$  that implements a certain broadcast abstraction  $B$  within the model  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$ . The argument is then generalized to the case where  $n > k + 1$  in the proof of the main theorem. This is achieved by observing that processes  $p_j$ , for  $j > k + 1$ , may fail at the beginning of the execution.

■ **Algorithm 1** Adversarial scheduler used by Definition 4.

---

```

1 Procedure adversarial_scheduler( $k, N, B, \mathcal{B}$ ) is:
2    $\alpha \leftarrow \varepsilon$ ;  $sent \leftarrow \emptyset$ ;  $decided \leftarrow [[\perp, \dots], \dots, [\perp, \dots]]$ ;
3   for  $i$  from 1 to  $k + 1$  do
4      $step \leftarrow \perp$ ;  $local\_del \leftarrow 0$ ;
5     while  $local\_del < N$  do
6       if  $step = \perp \vee step = \langle p_i : \text{return from } B.\text{sync-broadcast}(\text{SYNCH}) \rangle$  then
7          $step \leftarrow \langle p_i : B.\text{sync-broadcast}(\text{SYNCH}) \rangle$ ;
8       else  $step \leftarrow p_i$ 's next local step in  $C(\alpha)$ , according to  $\mathcal{B}$ ;
9        $\alpha \leftarrow \alpha \oplus step$ ;
10      if  $step = \langle p_i : \text{send } m \text{ to } p_i \rangle$  for some  $m$  then
11         $\alpha \leftarrow \alpha \oplus \langle p_i : \text{receive } m \text{ from } p_i \rangle$ 
12      else if  $step = \langle p_i : \text{send } m \text{ to } p_j \rangle$  for some  $m$  and  $j \neq i$  then
13         $sent.add(\langle m, i, j \rangle)$ 
14      else if  $step = \langle p_i : B.\text{deliver } m \text{ from } p_i \rangle$  for some  $m$  then
15         $local\_del \leftarrow local\_del + 1$ ;
16      else if  $step = \langle p_i : ksa.\text{propose}(v) \rangle$  for some  $k$ -SA object  $ksa$  and  $v$  then
17        if  $i = k + 1 \wedge \forall j \leq k : decided[ksa][j] \neq \perp$  then
18           $decided[ksa][i] \leftarrow decided[ksa][k]$ ;
19        else  $decided[ksa][i] \leftarrow v$ ;
20         $\alpha \leftarrow \alpha \oplus \langle p_i : ksa.\text{decide}(decided[ksa][i]) \rangle$ ;
21        if  $i = k \wedge \forall j \leq k : decided[ksa][j] \neq \perp$  then
22          foreach  $m : \langle m, k, k + 1 \rangle \in sent$  do
23             $\alpha \leftarrow \alpha \oplus \langle p_{k+1} : \text{receive } m \text{ from } p_k \rangle$ ;
24             $sent.remove(\langle m, k, k + 1 \rangle)$ ;
25           $local\_del \leftarrow -1$ ;
26      foreach  $\langle m, i, j \rangle \in sent$  do  $\alpha \leftarrow \alpha \oplus \langle p_j : \text{receive } m \text{ from } p_i \rangle$ ;
27      return  $\alpha$ ;
```

---

The adversarial execution  $\alpha_{k,N,B,\mathcal{B}}$  is constructed by an adversarial scheduler that follows the procedure outlined in Algorithm 1. The scheduler decides when processes get executed, when messages are received, and which values get decided upon by  $k$ -SA objects. It produces an execution admitted by the model  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  as validated by lemmas 5 to 12, though it does not necessarily exploit all possibilities offered by the specification of the model.

The scheduler begins with a sequential execution of all processes, ranging from  $p_1$  to  $p_{k+1}$ . During this phase, each process  $p_i$  repetitively calls  $B.\text{sync-broadcast}(\text{SYNCH})$ , where SYNCH represents a message with void content used for synchronization purpose, until it has  $B$ -delivered  $N$  of its own messages. This part of the execution remains indistinguishable to  $p_i$  from an execution  $\gamma_{k,N,B,\mathcal{B},i}$ , where other processes  $p_j$  would have crashed before the local delivery of their own  $N$  messages. To achieve this, processes decide on their own value on  $k$ -SA objects whenever possible, and the transmission of their messages to other processes is deferred by the scheduler until the end of this phase. However, a complication arises when all processes propose a value on the same  $k$ -SA object  $ksa$ . In such scenarios,  $p_{k+1}$  is compelled to decide on the value proposed by  $p_k$  to maintain the  $k$ -SA-AGREEMENT property. This decision renders  $p_{k+1}$ 's execution distinguishable from a scenario where  $p_k$

## 21:12 No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement

had initially crashed, allowing  $p_{k+1}$  to await  $p_k$ 's message. As a result, all messages sent by  $p_k$  to  $p_{k+1}$  are received by  $p_{k+1}$  (lines 22 to 24), and the messages that  $p_k$   $B$ -broadcast before this juncture are excluded from its count of  $N$  messages.

Subsequently, in a later phase of the algorithm, all processes receive all messages that were sent to them in the initial stage but have yet to be received, as delineated in Line 26. Algorithm 1 concludes by returning the execution halted at this juncture. Notably, at this point of termination, not all messages that have been  $B$ -broadcast are necessarily  $B$ -delivered by every process. However, this does not pose a problem for our analysis: the counterexample required for the proof in the following section involves a safety property that is already violated in the execution prefix returned by the algorithm. The scheduler maintains the following main variables:

- $\alpha$ , which is initially an empty sequence  $\varepsilon$ , is the execution currently being constructed.
- $i$ , which stores the identifier of the process currently under execution.
- $sent$ , initially set to  $\emptyset$ , is a set of triplets. A triplet  $\langle m, i, j \rangle$  is included in  $sent$  when a message  $m$  has been sent by process  $p_i$  to process  $p_j$ , but has not yet been received by  $p_j$ .
- $decided[ksa][j]$  is a two-dimensional associative array. The keys  $ksa$  correspond to  $k$ -SA objects used in  $\mathcal{B}$ , and  $j$  represents process identifiers. The values are either potential values that can be proposed to  $k$ -SA objects in  $\mathcal{B}$ , or a special value  $\perp$  that cannot be proposed. For each  $ksa$  and  $j$ ,  $decided[ksa][j]$  is initially set to  $\perp$ . It is later updated to value  $w$  when the process  $p_j$  decides on  $w$  for  $ksa$ .
- $local\_del$  tracks the number of messages that process  $p_i$   $B$ -delivers from itself, while avoiding communication with other processes. Under normal conditions,  $local\_del$  cycles through values from 0 to  $N - 1$  for each process  $p_i$ . However, if communication between processes  $p_k$  and  $p_{k+1}$  is inevitable during the execution of a  $B.sync\text{-broadcast}(m)$  operation by  $p_k$ ,  $local\_del$  is assigned a value of  $-1$ . This assignment signifies that  $local\_del$  will be reset to 0 once  $p_k$  completes its  $B$ -broadcast operation. Consequently, this setup enables  $p_k$  to  $B$ -deliver  $N$  of its own messages (excluding  $m$ ) without engaging in communication.
- $step$  identifies the subsequent step to be executed by Process  $p_i$ , represented either by the pair  $\langle p_i, action \rangle$  or by the special value  $\perp$  if the step is yet to be determined. In this context, there are two primary scenarios to consider. Firstly, if  $p_i$  has initiated the  $B.sync\text{-broadcast}$  operation but has not yet completed this invocation, then the deterministic algorithm  $B$  is responsible for defining the subsequent step that  $p_i$  must execute (Line 8). This step is crucial to fulfilling the BC-LOCAL-TERMINATION property of  $B$  within the configuration  $C(\alpha)$ , which delineates its local state after the execution  $\alpha$ . In the second scenario, if the aforementioned condition does not hold,  $p_i$  proceeds to  $B$ -broadcast a new message, specifically SYNCH.

Definition 4 now outlines the adversarial executions  $\alpha_{k,N,B,\mathcal{B}}$ ,  $\beta_{k,N,B,\mathcal{B}}$ , and  $\gamma_{k,N,B,\mathcal{B},i}$ . Our subsequent objective is to demonstrate that  $\alpha_{k,N,B,\mathcal{B}}$  qualifies as an admissible execution of the  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  model. It is required to verify that  $\alpha_{k,N,B,\mathcal{B}}$  is well-formed (as per Lemma 10), upholds the three defining properties of the  $k$ -set agreement:  $k$ -SA-VALIDITY (Lemma 5),  $k$ -SA-AGREEMENT (Lemma 6), and  $k$ -SA-TERMINATION (Lemma 7). and ensures compliance with the three properties of send/receive communication: SR-VALIDITY (Lemma 8), SR-NO-DUPLICATION (Lemma 9), and SR-TERMINATION (Lemma 12).

► **Definition 4** (Adversarial execution). *The following executions are defined:*

- $\alpha_{k,N,B,\mathcal{B}}$  is the execution produced by the procedure `adversarial_scheduler`( $k, N, B, \mathcal{B}$ ), as delineated in Algorithm 1.

- $\beta_{k,N,B,\mathcal{B}}$  constitutes a subsequence of  $\alpha_{k,N,B,\mathcal{B}}$ , encompassing only those steps that involve events associated with  $B$ . This includes the invocations of, or the responses from, the  $B$ .broadcast operation, as well as any  $B$ -delivery event.
  - For each  $i \in 1, \dots, k+1$ ,  $\gamma_{k,N,B,\mathcal{B},i}$  is derived from  $\alpha_{k,N,B,\mathcal{B}}$  by limiting it to, on the one hand, the steps of process  $p_i$  occurring strictly before Line 26; and on the other hand, the steps performed by  $p_k$  that are succeeded by a reset of `local_del` on Line 25. In these executions, all processes  $p_j \notin \{p_i, p_k\}$  are assumed to have crashed initially. Furthermore,  $p_k$  is treated as having crashed before executing its first step in  $\alpha_{k,N,B,\mathcal{B}}$  that is absent in  $\gamma_{k,N,B,\mathcal{B},i}$ , should such a step be present.
- **Lemma 5** ( $k$ -SA-Validity). In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process decides on a value  $w$  on a  $k$ -SA object  $ksa$ , then the value  $w$  was proposed by some process on  $ksa$ .
  - **Lemma 6** ( $k$ -SA-Agreement). In both  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  executions, no more than  $k$  distinct values are decided on any given  $k$ -SA object.
  - **Lemma 7** ( $k$ -SA-Termination). In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process proposes a value on a  $k$ -SA object  $ksa$ , then this process will also decide a value on  $ksa$ .
  - **Lemma 8** (SR-Validity). In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process  $p_r$  receives a message  $m$  from process  $p_s$ , then process  $p_s$  has indeed sent  $m$  to  $p_r$ .
  - **Lemma 9** (SR-No-Duplication). In both  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  executions, each message is received at most once.
  - **Lemma 10** (Well-Formed Executions).  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  are well-formed executions of  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  with respect to  $\mathcal{B}$ .
  - **Lemma 11** (Termination of Algorithm 1). The execution  $\alpha_{k,N,B,\mathcal{B}}$  is finite.
  - **Lemma 12** (SR-Termination). In  $\alpha_{k,N,B,\mathcal{B}}^1$ , if a process  $p_s$  sends a message  $m$  to a correct process  $p_r$ , then  $p_r$  will eventually receive  $m$  from  $p_s$ .

### 4.3 $N$ -Solo Executions and the Contradiction

► **Definition 13** ( $N$ -solo executions). Let  $\beta$  be an execution of the model  $\mathcal{CAMP}_n[B]$ , and let  $N \in \mathbb{N}$ . We say that  $\beta$  is  $N$ -solo if there exists a set  $S$  of messages  $B$ -broadcast in  $\beta$  that meets the following conditions:

- $S$  contains exactly  $N$  messages  $B$ -broadcast by each process  $p_i$ , hereafter referred to as  $m_{i,1}, \dots, m_{i,N}$ .
- Each process  $p_i$   $B$ -delivers all its own messages in  $S$  before any other message in  $S$ .

► **Lemma 14.** For all  $k > 1$  and  $N > 0$ , if there exists an algorithm  $\mathcal{B}$  that implements some broadcast abstraction  $B$  in the model  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$ , then  $B$  admits an  $N$ -solo execution.

**Proof.** Assume  $k > 1$  and  $N > 0$ , and suppose an algorithm  $\mathcal{B}$  implements a broadcast abstraction  $B$  in  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$ . According to Lemmas 5 to 12,  $\alpha_{k,N,B,\mathcal{B}}$  constitutes an admissible  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  execution, thus by  $\mathcal{B}$ 's correctness,  $\beta_{k,N,B,\mathcal{B}}$  is admitted by  $B$ . We aim to demonstrate that  $\beta_{k,N,B,\mathcal{B}}$  is  $N$ -solo. For each  $i \in \{1, \dots, k+1\}$ , the loop starting on Line 5 halts by Lemma 11, but only after `local_del` has been incremented at least  $N$  times

<sup>1</sup> Unlike previous lemmas, this property may not hold for  $\gamma_{k,N,B,\mathcal{B},i}$  in the general case.

on Line 15, without having been reset on Line 25. Each of these increments corresponds to the  $B$ -delivery, by  $p_i$ , of its own message  $m_{i,local\_del}$ . We now prove that these  $(k + 1) \times N$  messages satisfy the criteria in Definition 13.

Consider two distinct processes  $p_i$  and  $p_j$ , assuming without loss of generality that  $i < j$ . Due to the sequential nature of the loop on Line 3,  $p_i$   $B$ -delivers all its own messages before  $p_j$  even begins its  $B$ -broadcasts. Consequently, by the BC-VALIDITY property of  $B$ ,  $p_i$  completes delivering its messages before any of  $p_j$ 's. Lemmas 5 to 10 confirm that  $\gamma_{k,N,B,B,j}$  upholds all safety properties of send/receive and  $k$ -SA objects, and is well-formed, indicating  $\gamma_{k,N,B,B,j}$  is the prefix of an execution of  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$ . In  $\gamma_{k,N,B,B,j}$ ,  $p_i$  does not  $B$ -broadcast its messages  $m_{i,1}, \dots, m_{i,N}$ . This is because, for  $i <$ ,  $p_i$  does not take any step in  $\gamma_{k,N,B,B,j}$  by the definition of  $\gamma_{k,N,B,B,j}$ , and if  $i = k$  and  $j = k + 1$ , all messages  $m_{i,1}, \dots, m_{i,N}$  are broadcast after a reset of  $local\_del$  in  $\alpha_{k,N,B,B}$  to satisfy the condition of Line 5. Hence their broadcast events are excluded from  $\gamma_{k,N,B,B,j}$ , so  $p_j$  does not  $B$ -deliver these messages, as ensured by  $B$ 's correctness and BC-VALIDITY of  $B$ . Since  $\alpha_{k,N,B,B}$  and  $\gamma_{k,N,B,B,j}$  share identical  $p_j$  steps before Line 26, in  $\alpha_{k,N,B,B}$ ,  $p_j$   $B$ -delivers all its own messages before Line 26, without  $B$ -delivering any of the messages of  $p_i$ . Consequently,  $\beta_{k,N,B,B}$ , which includes only  $B$ -related steps from  $\alpha_{k,N,B,B}$ , is an  $N$ -solo execution admitted by  $B$ . ◀

► **Lemma 15.** *For all  $k > 1$ , and for every content-neutral and compositional broadcast abstraction  $B$ , if there exists an algorithm  $\mathcal{A}$  that solves  $k$ -SA in the model  $\mathcal{CAMP}_{k+1}[B]$ , then there exists an integer  $N > 0$  such that  $B$  does not allow any  $N$ -solo execution.*

**Proof.** Assume  $B$  is a broadcast abstraction and  $\mathcal{A}$  is an algorithm solving  $k$ -SA in the model  $\mathcal{CAMP}_{k+1}[B]$ . It's noteworthy that  $\mathcal{A}$  can be transformed into an alternative algorithm,  $\mathcal{A}'$ , which also solves  $k$ -SA in the same model but without relying on the point-to-point primitives **send** and **receive**. This transformation is feasible because the **send** and **receive** primitives can be trivially emulated using  $B$ . Moreover, the correctness of  $\mathcal{A}'$  results from the compositionality of  $B$ . Specifically, the executions of  $\mathcal{A}'$ , when projected onto the set of messages shared with  $\mathcal{A}$  (excluding those utilized solely for simulating **send/receive** in  $\mathcal{A}'$ ), are admitted by  $\mathcal{CAMP}_{k+1}[B]$ , thereby yielding identical results in  $\mathcal{A}$  and  $\mathcal{A}'$ .

Consider an execution  $\alpha_i$  where a process  $p_i \in \Pi$  proposes  $i$  to a  $k$ -SA object using  $\mathcal{A}'$ , while all other processes crash before taking any step. Due to the  $k$ -SA-TERMINATION property of the  $k$ -SA object,  $p_i$  eventually decides on a value. The  $k$ -SA-VALIDITY property ensures this value is  $i$ . Denote by  $m_{i,1}, \dots, m_{i,N_i}$  the sequence of messages  $p_i$   $B$ -delivers in  $\alpha_i$  prior to its decision.

Let  $N = \max\{1, N_1, \dots, N_{k+1}\}$ , and suppose  $B$  admits an  $N$ -solo execution  $\beta$ . Construct  $\gamma$  as the sub-execution of  $\beta$  restricted to the message contained in the set  $S$  given by Definition 13. Due to the compositionality of  $B$ ,  $\gamma$  is an execution admitted by  $B$ , where each process  $p_i$   $B$ -delivers its  $N_i$  messages before any message from other processes. Now, define  $\delta$  from  $\gamma$  by replacing each process  $p_i$ 's  $N_i$  messages with the messages  $m_{i,1}, \dots, m_{i,N_i}$  from  $\alpha_i$ . The content-neutrality of  $B$  ensures that  $\delta$  is admitted by  $B$ . For each process  $p_i$ ,  $\alpha_i$  is indistinguishable from  $\delta$ , as both executions involve identical  $B$ -broadcast and  $B$ -delivery steps for  $p_i$ . Hence, when  $\mathcal{A}'$  is executed on  $\delta$ , each  $p_i$  decides on its own value  $i$ , leading to  $k + 1$  distinct decisions. This contradicts the  $k$ -SA-AGREEMENT property of  $k$ -SA. Therefore, such  $\beta$  cannot exist, implying  $B$  does not allow any  $N$ -solo execution. ◀

► **Theorem 16.** *For all  $n, k$  such that  $1 < k < n$ , there is no content-neutral and compositional broadcast abstraction equivalent to  $k$ -SA in a crash-prone asynchronous message-passing system where any number of processes may crash.*

► **Corollary 17.** *For all  $n, k$  such that  $1 < k < n$ , it is impossible to implement  $k$ -BO Broadcast in the model  $\mathcal{CAMP}_n[k\text{-SA}]$ , without restrictions on the number of failures.*

## 5 Conclusion

This paper investigates the computational equivalence of any broadcast abstraction to  $k$ -set agreement ( $k$ -SA) in message-passing systems. Following the introduction of two new symmetry properties defining admissible broadcast abstractions – *compositionality* and *content-neutrality* – we demonstrate that no broadcast abstraction, which is both content-neutral and compositional, is computationally equivalent to  $k$ -set agreement when  $1 < k < n$ . This paper highlights a crucial distinction in the application of  $k$ -set agreement in shared memory versus message-passing systems: for  $1 < k < n$ ,  $k$ -SA is equivalent to a broadcast abstraction in shared memory (specifically,  $k$ -BO broadcast), but no such equivalence exists in message-passing systems.

As Lamport famously observed in [15], “The concept of time (...) is derived from the more fundamental concept of the order in which events occur.” Therefore, at the abstraction level of message broadcasting in the system, each broadcast abstraction inherently provides a definition of time. On one end of the spectrum, broadcast abstractions that can be implemented solely through send and receive operations, such as Causal broadcast, offer processes a relativistic notion of time, defined by the “happened before” relation – a partial order. Conversely, at the other extreme where processes can utilize consensus, the set of broadcast events in Total Order broadcast forms an absolute timeline, known to all processes. Under this interpretation,  $k$ -SA represents a symmetric predicate on time – hence an elegant synchronization problem – when utilized within a shared-memory model, but not in message-passing systems.

This non-existence result questions the usefulness of  $k$ -SA in message-passing systems. Indeed, the absence of a compositional (and content-neutral) broadcast abstraction arises from algorithmic challenges in the orchestration of a composition of  $k$ -SA objects within a single algorithm. To illustrate these challenges, consider a full-information algorithm operating within the  $\mathcal{CAMP}_n[k\text{-SA}]$  model, where processes alternate between sending messages and proposing their full local knowledge to a sequence of  $k$ -SA objects. After deciding on the first  $k$ -SA object, the system is left with at most  $k$  competing perspectives. Consequently, at most  $k$  distinct values are proposed to the following  $k$ -SA object, allowing each process to decide on its own proposed value. This outcome renders all subsequent  $k$ -SA objects redundant, as they do not force any further synchronization. Hence, the introduction of several instances of the task does not reduce system uncertainty more effectively than a single instance.

This situation contrasts with the dynamics in shared memory systems, where alternating agreement phases through  $k$ -SA instances and coordination phases through shared registers helps create a sequence of asynchronous rounds. This effectively allows “weaving” the  $k$ -SA objects within the temporal framework provided by the memory, lending purpose to their iterative composition. Nonetheless, the existence of  $k$ -BO Broadcast, which is not stronger than an atomic register, suggests that shared memory is not necessary for such weaving. This observation opens the perspective for identifying the weakest temporal properties that effectively allow the characterization of  $k$ -set-agreement by a symmetric broadcast abstraction.

## References

- 1 Yehuda Afek, Eli Gafni, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. The  $k$ -simultaneous consensus problem. *Distributed Comput.*, 22(3):185–195, 2010. doi:10.1007/S00446-009-0090-8.
- 2 Mustaque Ahamad, Gil Neiger, James E Burns, Prince Kohli, and Phillip W Hutto. Causal memory: Definitions, implementation, and programming. *Distributed Computing*, 9:37–49, 1995. doi:10.1007/BF01784241.
- 3 Kenneth P Birman and Thomas A Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems (TOCS)*, 5(1):47–76, 1987. doi:10.1145/7351.7478.
- 4 Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for  $t$ -resilient asynchronous computations. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 91–100. ACM, 1993. doi:10.1145/167088.167119.
- 5 Zohir Bouzid and Corentin Travers. Parallel consensus is harder than set agreement in message passing. In *IEEE 33rd International Conference on Distributed Computing Systems, ICDCS 2013, 8-11 July, 2013, Philadelphia, Pennsylvania, USA*, pages 611–620, 2013. doi:10.1109/ICDCS.2013.72.
- 6 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi:10.1145/226643.226647.
- 7 Soma Chaudhuri. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993. doi:10.1006/INCO.1993.1043.
- 8 Mathilde Déprés, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Brief announcement: The mbroadcast abstraction. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 282–285. ACM, 2023. doi:10.1145/3583668.3594569.
- 9 Mathilde Déprés, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Send/receive patterns versus read/write patterns in crash-prone asynchronous distributed systems. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPICs*, pages 16:1–16:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.DISC.2023.16.
- 10 Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2), 1985. doi:10.1145/3149.214121.
- 11 Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report Tech Report 94-1425, Cornell University, 1994. Extended version of "Fault-Tolerant Broadcasts and Related Problems" in *Distributed systems, 2nd Edition*, Addison-Wesley/ACM, pp. 97-145 (1993).
- 12 Maurice Herlihy and Nir Shavit. The asynchronous computability theorem for  $t$ -resilient tasks. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 111–120. ACM, 1993. doi:10.1145/167088.167125.
- 13 Damien Imbs, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Which broadcast abstraction captures  $k$ -set agreement? In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPICs*, pages 27:1–27:16, 2017. doi:10.4230/LIPICS.DISC.2017.27.
- 14 Damien Imbs, Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. Set-constrained delivery broadcast: A communication abstraction for read/write implementable distributed objects. *Theoretical Computer Science*, 886:49–68, 2021. doi:10.1016/J.TCS.2021.06.044.
- 15 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications*, 1978.
- 16 Leslie Lamport. Generalized consensus and paxos, 2005.



- 17 Fernando Pedone and André Schiper. Generic broadcast. In *Distributed Computing: 13th International Symposium, DISC'99 Bratislava, Slovak Republic September 27–29, 1999 Proceedings 13*, pages 94–106. Springer, 1999.
- 18 David Powell. Group communication (introduction to the special section). *Commun. ACM*, 39(4):50–53, 1996. doi:10.1145/227210.227225.
- 19 Michel Raynal. Set agreement. In *Encyclopedia of Algorithms*, pages 1956–1959. Springer, 2016. doi:10.1007/978-1-4939-2864-4\_367.
- 20 Michel Raynal. *Fault-tolerant message-passing distributed systems: an algorithmic approach*. Springer, 2018. doi:10.1007/978-3-319-94141-7.
- 21 Michel Raynal, André Schiper, and Sam Toueg. The causal ordering abstraction and a simple way to implement it. *Information processing letters*, 39(6):343–350, 1991. doi:10.1016/0020-0190(91)90008-6.
- 22 Michael E. Saks and Fotios Zaharoglou. Wait-free  $k$ -set agreement is impossible: the topology of public knowledge. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 101–110. ACM, 1993. doi:10.1145/167088.167122.
- 23 Fred B. Schneider. The state machine approach: A tutorial. In *Proc. of Asilomar Workshop on Fault-Tolerant Distributed Computing*, volume 448 of *LNCS*, pages 18–41. Springer, 1986. doi:10.1007/BFB0042323.
- 24 Tim Wu. Network neutrality, broadband discrimination. *J. on Telecomm. & High Tech. L.*, 2:141, 2003.

## A Proofs of Lemmas

► **Lemma 5** (*k*-SA-Validity). *In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process decides on a value  $w$  on a  $k$ -SA object  $ksa$ , then the value  $w$  was proposed by some process on  $ksa$ .*

**Proof.** Assume that  $\alpha_{k,N,B,\mathcal{B}}$  includes a step  $\langle p_j : ksa.decide(w) \rangle$ . This step originates from Line 20, following  $p_j$ 's invocation of  $ksa.propose(v)$ . Consequently,  $w = decided[ksa][j]$ , that was set either on Line 18 or Line 19.

- If  $decided[ksa][j]$  was assigned on Line 19, then  $w = v$ . The step  $\langle p_j : ksa.propose(v) \rangle$  would have been included in  $\alpha_{k,N,B,\mathcal{B}}$  at Line 9.
- Otherwise,  $w = decided[ksa][k]$ , and per Line 17,  $i = k + 1$ . In this case,  $decided[ksa][k] \neq \perp$  was previously set by  $p_k$  in  $\alpha_{k,N,B,\mathcal{B}}$  on Line 19, following the inclusion of the step  $\langle p_k : ksa.propose(w) \rangle$  in  $\alpha_{k,N,B,\mathcal{B}}$ .

This sequence of events establishes the property for  $\alpha_{k,N,B,\mathcal{B}}$ . Consider now the case of  $\gamma_{k,N,B,\mathcal{B},i}$  containing a step  $\langle p_j : ksa.decide(w) \rangle$ , following the same case disjunction as before. In the case of Line 19, the property holds because  $\gamma_{k,N,B,\mathcal{B},i}$  and  $\alpha_{k,N,B,\mathcal{B}}$  both encompass identical **propose** and **decide** steps executed by  $p_j$ . In the second case, the fulfillment of the condition at Line 21 for  $p_k$  leads to the subsequent reset of  $local\_del$  on Line 25. Therefore, in both cases, the step  $\langle p_k : ksa.propose(w) \rangle$  is also included in  $\gamma_{k,N,B,\mathcal{B},i}$ . ◀

► **Lemma 6** (*k*-SA-Agreement). *In both  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  executions, no more than  $k$  distinct values are decided on any given  $k$ -SA object.*

**Proof.** By the definition of  $\gamma_{k,N,B,\mathcal{B},i}$ , at most two processes, specifically  $p_i$  and  $p_k$ , are capable of deciding a value in  $\gamma_{k,N,B,\mathcal{B},i}$ , satisfying the condition as  $2 \leq k$ .

Assume that in  $\alpha_{k,N,B,\mathcal{B}}$ ,  $k + 1$  distinct values are decided. Given that processes execute sequentially, processes  $p_1$  through  $p_k$  would have already recorded their value in  $decided[ksa][.]$  before  $p_{k+1}$  proposing its value. Consequently, the condition at Line 17 would be met, leading to  $p_{k+1}$  deciding the same value as  $p_k$ , thus resulting in a contradiction. ◀

## 21:18 No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement

► **Lemma 7** ( $k$ -SA-Termination). *In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process proposes a value on a  $k$ -SA object  $ksa$ , then this process will also decide a value on  $ksa$ .*

**Proof.** Suppose that  $\alpha_{k,N,B,\mathcal{B}}$  includes a step  $\langle p_j : ksa.\text{propose}(v) \rangle$ . This step was introduced on Line 9. Subsequently, the condition at Line 16 is satisfied, leading to the inclusion of a step  $\langle p_j : ksa.\text{decide}(w) \rangle$  in  $\alpha_{k,N,B,\mathcal{B}}$  at Line 20. This confirms the lemma for  $\alpha_{k,N,B,\mathcal{B}}$ .

Now, assume  $\gamma_{k,N,B,\mathcal{B},i}$  contains a step  $\langle p_j : ksa.\text{propose}(v) \rangle$ . Here,  $j$  can only be either  $i$  or  $k$ .

- If  $j = i$ , then  $\gamma_{k,N,B,\mathcal{B},i}$  includes the same step  $\langle p_j : ksa.\text{decide}(w) \rangle$  as found in  $\alpha_{k,N,B,\mathcal{B}}$ .
- If  $j = k$ , it is important to note that the steps  $\langle p_j : ksa.\text{propose}(v) \rangle$  (at Line 9) and  $\langle p_j : ksa.\text{decide}(w) \rangle$  (at Line 20) cannot be isolated by a reset of  $local\_del$  on Line 25.

Therefore, if the proposal step exists in  $\gamma_{k,N,B,\mathcal{B},i}$ , the decision step must also be present. In both scenarios, the lemma's condition is satisfied in  $\gamma_{k,N,B,\mathcal{B},i}$ , thus completing the proof. ◀

► **Lemma 8** (SR-Validity). *In the executions  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$ , if a process  $p_r$  receives a message  $m$  from process  $p_s$ , then process  $p_s$  has indeed sent  $m$  to  $p_r$ .*

**Proof.** Assume that  $\alpha_{k,N,B,\mathcal{B}}$  includes a step  $\langle p_r : \text{receive } m \text{ from } p_s \rangle$ . This step is either introduced on Line 11 following a step  $\langle p_s : \text{send } m \text{ to } p_r \rangle$  where  $r = s$ , or on Line 23 or Line 26 when  $\langle m, s, r \rangle \in sent$ . The triplet  $\langle m, s, r \rangle$  is added to  $sent$  only on Line 13, implying that  $\langle p_s : \text{send } m \text{ to } p_r \rangle$  was previously included in  $\alpha_{k,N,B,\mathcal{B}}$  on Line 9. This confirms the lemma for  $\alpha_{k,N,B,\mathcal{B}}$ .

Now, consider a receive step in  $\gamma_{k,N,B,\mathcal{B},i}$ . Given the previous argument,  $\alpha_{k,N,B,\mathcal{B}}$  must contain a corresponding send step. Since receive steps from Line 26 are not part of  $\gamma_{k,N,B,\mathcal{B},i}$ , there are two possible scenarios:

- If the receive step is added to  $\gamma_{k,N,B,\mathcal{B},i}$  on Line 11, then the preceding send step is also included in  $\gamma_{k,N,B,\mathcal{B},i}$ .
- If the receive step is added to  $\gamma_{k,N,B,\mathcal{B},i}$  on Line 23, the sender is  $p_k$ , and  $local\_del$  was reset on Line 25 subsequently. Therefore, the send step is also present in  $\gamma_{k,N,B,\mathcal{B},i}$ .

Both cases confirm the lemma's condition on  $\gamma_{k,N,B,\mathcal{B},i}$ , thus completing the proof. ◀

► **Lemma 9** (SR-No-Duplication). *In both  $\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  executions, each message is received at most once.*

**Proof.** The property for  $\alpha_{k,N,B,\mathcal{B}}$  is substantiated by the message reception mechanics: a message can only be received on Line 11, in which case it is not added to  $sent$  so it is not received again later, on Line 23 followed by its removal from  $sent$ , or singularly on Line 26 due to  $sent$ 's set semantics. Since  $\gamma_{k,N,B,\mathcal{B},i}$  comprises only a subset of  $\alpha_{k,N,B,\mathcal{B}}$ 's reception events, the lemma is valid for  $\gamma_{k,N,B,\mathcal{B},i}$  as well. ◀

► **Lemma 10** (Well-Formed Executions).  *$\alpha_{k,N,B,\mathcal{B}}$  and  $\gamma_{k,N,B,\mathcal{B},i}$  are well-formed executions of  $\mathcal{CAMP}_{k+1}[k\text{-SA}]$  with respect to  $\mathcal{B}$ .*

**Proof.** To validate the property for  $\alpha_{k,N,B,\mathcal{B}}$ , we observe that the participation of only processes  $p_1$  to  $p_{k+1}$  stems for (1) loop bounds defined on Line 7, and (2) the SR-VALIDITY property and the correctness of  $\mathcal{B}$  for the receiving processes on Line 26. A process initiates the operation  $B.\text{broadcast}$  either at the start of its execution on Line 4, or immediately after returning from its previous invocation, as indicated on Lines 6 and 7. This ensures adherence to the required pattern of alternating invocations and responses. Furthermore, the sequence of steps a process follows between its invocations and responses is consistent with  $\mathcal{B}$ , as defined on Line 8.

As for  $\gamma_{k,N,B,\mathcal{B},i}$ , the property comes from the fact that for all processes  $p_j$ , the sequence of steps taken by  $p_j$  in  $\gamma_{k,N,B,\mathcal{B},i}$  is a prefix of the sequence of steps taken by  $p_j$  in  $\alpha_{k,N,B,\mathcal{B}}$ . ◀

► **Lemma 11** (Termination of Algorithm 1). *The execution  $\alpha_{k,N,B,\mathcal{B}}$  is finite.*

**Proof.** Assume for contradiction that  $\alpha_{k,N,B,\mathcal{B}}$  contains an infinite number of steps. Given that Algorithm 1 includes no recursion and only one while loop, there exists some  $i \in \{1, \dots, k+1\}$  engaged in an infinite loop starting at Line 5 with  $local\_del < N$  remaining true indefinitely.

By Lemmas 5 to 9,  $\gamma_{k,N,B,\mathcal{B},i}$  satisfies all the conditions required for an admissible execution, except SR-TERMINATION. Let us establish that  $\gamma_{k,N,B,\mathcal{B},i}$  also verifies SR-TERMINATION:

- For  $i < k$ ,  $\gamma_{k,N,B,\mathcal{B},i}$  contains only messages sent by  $p_i$ , as the  $i^{th}$  iteration does not terminate. Process  $p_i$  receives its own messages on Line 11, and others are not required to receive them as they have crashed.
- For  $i = k$ , similar to the previous case,  $\gamma_{k,N,B,\mathcal{B},i}$  includes only messages by  $p_i$  by definition of  $\gamma_{k,N,B,\mathcal{B},i}$ . Message reception follows the same logic as above.
- For  $i = k+1$ , note that  $p_k$  is considered faulty in  $\gamma_{k,N,B,\mathcal{B},i}$  due to (1) taking a finite number of steps in  $\alpha_{k,N,B,\mathcal{B}}$  since  $p_i$  is executed after  $p_k$ 's last step, and (2) the condition  $local\_del < N$  only becoming false post Line 15 which is preceded by a step  $\langle p_k : B.deliver\ m\ from\ p_k \rangle$  that belongs to  $\alpha_{k,N,B,\mathcal{B}}$  but not  $\gamma_{k,N,B,\mathcal{B},i}$ . Therefore, it suffices to show that  $p_i$  receives all messages directed to it. Only  $p_k$  and  $p_i$  send messages in  $\gamma_{k,N,B,\mathcal{B},i}$ . Process  $p_i$  receives its own messages on Line 11, and all messages sent by  $p_k$  to  $p_i$  in  $\gamma_{k,N,B,\mathcal{B},i}$  are sent prior to the reset of  $local\_del$ , hence they are received by  $p_i$  on Line 23.

Therefore,  $\gamma_{k,N,B,\mathcal{B},i}$  is an execution admitted by the model  $\mathcal{CAM}\mathcal{P}_{k+1}[k\text{-SA}]$ , in which  $p_i$  takes an infinite number of steps. By correctness of  $\mathcal{B}$  and the BC-GLOBAL-CS-TERMINATION property of  $B$ , all  $B$ -broadcast messages by  $p_i$  in  $\gamma_{k,N,B,\mathcal{B},i}$  must eventually be  $B$ -delivered by  $p_i$  in  $\gamma_{k,N,B,\mathcal{B},i}$ . Moreover, since  $\gamma_{k,N,B,\mathcal{B},i}$  and  $\alpha_{k,N,B,\mathcal{B}}$  contain the same steps of  $p_i$ , all messages  $B$ -broadcast by  $p_i$  in  $\alpha_{k,N,B,\mathcal{B}}$  are eventually  $B$ -delivered by  $p_i$  in  $\alpha_{k,N,B,\mathcal{B}}$ . Since  $p_i$  immediately  $B$ -broadcasts a new message after returning from its previous  $B.sync\text{-broadcast}$  invocation (Lines 6 to 7),  $p_i$   $B$ -delivers an infinite number of messages from itself, and repeatedly increments  $local\_del$  on Line 15. As  $local\_del$  is bounded by  $N$ , it must be reset on Line 25 infinitely, following proposals to  $k$ -SA objects.

Let  $K$  be the set of  $k$ -SA objects such that  $p_i$  executes Line 25 after proposing a value to them. Given the one-time proposal limit per  $k$ -SA object,  $K$  is infinite. Therefore, based on Line 21,  $i = k$ , and there is an infinite number of  $ksa$  such that  $decided[ksa][1] \neq \perp$ . However,  $decided[ksa][1]$  is set during the first iteration for an infinite number of distinct  $k$ -SA objects. This indicates that the first iteration does not terminate. This is a contradiction because (1)  $k > 1$  so  $k \neq 1$  and (2) the  $k^{th}$  iteration of the loop started because  $p_k$  takes (an infinite number of) steps in  $\alpha_{k,N,B,\mathcal{B}}$ . This contradiction implies that  $\alpha_{k,N,B,\mathcal{B}}$  must be finite, completing the proof. ◀

► **Lemma 12** (SR-Termination). *In  $\alpha_{k,N,B,\mathcal{B}}^2$ , if a process  $p_s$  sends a message  $m$  to a correct process  $p_r$ , then  $p_r$  will eventually receive  $m$  from  $p_s$ .*

**Proof.** Consider a message  $m$  sent by  $p_s$  to  $p_r$  in  $\alpha_{k,N,B,\mathcal{B}}$ . A step  $\langle p_s : send\ m\ to\ p_r \rangle$  is recorded in  $\alpha_{k,N,B,\mathcal{B}}$  at Line 9. If  $s = r$ , then a step  $\langle p_r : receive\ m\ from\ p_s \rangle$  is subsequently appended to  $\alpha_{k,N,B,\mathcal{B}}$  at Line 11. In contrast, if  $s \neq r$ ,  $\langle m, s, r \rangle$  is added to  $sent$  at Line 13.

<sup>2</sup> Unlike previous lemmas, this property is not proven for  $\gamma_{k,N,B,\mathcal{B},i}$  in the general case.

## 21:20 No Symmetric Broadcast Abstraction Characterizes $k$ -Set-Agreement

As established in Lemma 11,  $\alpha_{k,N,B,\mathcal{B}}$  is finite. If  $\langle m, s, r \rangle$  remains in *sent* at the conclusion of the execution, then a step  $\langle p_r : \text{receive } m \text{ from } p_s \rangle$  is appended to  $\alpha_{k,N,B,\mathcal{B}}$  at Line 26. Conversely, if  $\langle m, s, r \rangle$  is not present in *sent*, it implies that it was removed at Line 24 subsequent to appending a step  $\langle p_r : \text{receive } m \text{ from } p_s \rangle$  to  $\alpha_{k,N,B,\mathcal{B}}$  at Line 23. Therefore, in every case,  $p_r$  receives  $m$  from  $p_s$ . ◀

► **Theorem 16.** *For all  $n, k$  such that  $1 < k < n$ , there is no content-neutral and compositional broadcast abstraction equivalent to  $k$ -SA in a crash-prone asynchronous message-passing system where any number of processes may crash.*

**Proof.** Assume the existence of a content-neutral and compositional broadcast abstraction  $B$  that is equivalent to  $k$ -SA in the specified model. Let  $\mathcal{A}$  be an algorithm implementing  $k$ -SA in  $\mathcal{CAMP}_n[B]$ , and  $\mathcal{B}$  be an algorithm implementing  $B$  in  $\mathcal{CAMP}_n[k\text{-SA}]$ . Remark that the model  $\mathcal{CAMP}_n[\emptyset]$  is functionally identical to the model  $\mathcal{CAMP}_{k+1}[\emptyset]$  when  $n - k - 1$  processes crash at the start of execution. Hence, the two algorithms would still be correct in the model  $\mathcal{CAMP}_{k+1}[\emptyset]$ . By Lemma 15, there exists an integer  $N > 0$  such that  $B$  does not admit any  $N$ -solo execution. Conversely, by Lemma 14,  $B$  admits an  $N$ -solo execution. This contradiction implies the non-existence of such a broadcast abstraction  $B$ . ◀

► **Corollary 17.** *For all  $n, k$  such that  $1 < k < n$ , it is impossible to implement  $k$ -BO Broadcast in the model  $\mathcal{CAMP}_n[k\text{-SA}]$ .*

**Proof.** As established in Section 3,  $k$ -BO Broadcast is content-neutral and compositional. It has also been demonstrated in [13] that  $k$ -SA can be solved in the model  $\mathcal{CAMP}_n[k\text{-BO}]$ . Consequently, an algorithm that implements  $k$ -BO Broadcast in the model  $\mathcal{CAMP}_n[k\text{-SA}]$  would render the two abstractions equivalent, thereby contradicting Theorem 16. ◀