FAKULTÄT FÜR !NFORMATIK

Faculty of Informatics

Easy Impossibility Proofs for k-Set Agreement

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Kyrill Winkler, BSc.
Matrikelnummer 0201623

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Ulrich Schmid

Wien, 8. Oktober 2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

Easy Impossibility Proofs for k-Set Agreement

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Kyrill Winkler, BSc.
Registration Number 0201623

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Univ.Prof. Dr. Ulrich Schmid

Vienna, 8. Oktober 2013 ____________________ ____________________
(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Kyrill Winkler, BSc.
Ybbsstraße 27/17, 1020 Wien

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First and foremost, I would like to thank my supervisor, Prof. Ulrich Schmid for sparking my interest in the field of distributed computing as well as for his helpful advice and invaluable insights. Moreover, I would like to thank Manfred Schwarz, BSc., for the multitude of inspiring discussions we had during the creation of this thesis. In addition, I would like to thank both of them as well as Dr. Martin Biely and Dr. Peter Robinson for their collaboration on the topic of directed dynamic networks, which lead to the current version of Chapter 4. The support by the Austrian Science Fund (FWF) project RiSE (S11405) is also gratefully acknowledged.

Finally, I would like to thank my friends and family for their love and support. I would especially like to thank my parents and grandparents for providing me with the means to pursue an academic study. Last, but certainly not least, I would like to thank Angela Ladurner for her inexhaustible love, support, and patience.

Abstract

This thesis is concerned with impossibility results, i.e., proofs of the fact that certain classes of algorithms cannot exist. The algorithms investigated are from the field of fault-tolerant distributed computing, which is devoted to the formal study of processing entities, modeled as communicating state machines, that may possibly fail and communicate with each other by either exchanging messages or via access to a shared memory.

We investigate the problem of k-set agreement, a natural generalization of consensus. While consensus concerns itself with the task in which all processes eventually have to decide on a common value that was originally some process' input value, k-set agreement allows up to k different decision values. Hence, for $k = 1$, k-set agreement is equivalent to consensus. Although there exist impossibility results for deterministic consensus in systems prone to failures, relying solely on combinatoric arguments that might be considered classical today, the corresponding impossibility results for k-set agreement require complex arguments from algebraic topology. Nevertheless, there has been recent research on finding "easy" or non-topological impossibility proofs for k-set agreement, which may also provide a new handle on solving some long-standing open problems like the weakest failure detector for k-set agreement in message-passing systems.

The focus of this thesis lies on such non-topological impossibilities for k-set agreement. We present and discuss existing approaches and results and provide rigorous proofs for new results regarding various models and scenarios, including the important class of dynamic systems that may evolve over time.

This work has been supported by the Austrian Science Fund (FWF) project RiSE (S11405).

Kurzfassung

Diese Masterarbeit beschäftigt sich mit Resultaten, die beweisen, dass bestimmte Klassen von Algorithmen nicht existieren können. Sie ist im Umfeld der verteilten Algorithmen angesiedelt. Diese Disziplin behandelt die mathematisch-formale Untersuchung von kommunizierenden Zustandsautomaten, die als Prozesse bezeichnen werden. Diese Prozesse interagieren über das Austauschen von Nachrichten oder über einen gemeinsam verwendeten Speicher miteinander und können potentiell fehlerhaft sein.

Wir untersuchen das Problem des k-Set Agreement, einer Generalisierung des Consensus-Problems. Während die Prozesse beim Consensus-Problem die Aufgabe haben, sich schlussendlich auf einen einzigen Wert zu einigen, welcher der Eingangswert eines Prozesses war, sind bei k-Set Agreement systemweit k verschiedene Entscheidungen erlaubt. Im Falle von $k = 1$ ist k-Set Agreement äquivalent zu Consensus. Obwohl es Resultate gibt, die beweisen, dass in bestimmten fehlerhaften Systemen Consensus von keinem deterministischen Algorithmus gelöst werden kann, die man heutzutage wohl als "klassische" Resultate bezeichnen könnte und die auf rein kombinatorischen Methoden beruhen, benötigen die korrespondierenden Resultate für k-Set Agreement komplexere Argumente aus dem Bereich der algebraischen Topologie. Allerdings gibt es aktuelle Forschung in diesem Bereich, die nach "einfachen", d.h. nicht-topologischen, Beweisen für die Unmöglichkeit, k-Set Agreement zu lösen, sucht. Diese liefert möglicherweise auch einen neuen Ansatz, um noch ungelöste Probleme, wie etwa das Auffinden des schwächsten Failure Detectors für k-set agreement in Message-Passing Systemen, anzugehen.

Das Hauptaugenmerk dieser Arbeit liegt auf solchen nicht-topologischen Beweisen. Wir präsentieren und diskutieren existierende Herangehensweisen und liefern rigorose Beweise für neue Resultate für verschiedenste Modelle und Szenarios, u.a. für die wichtige Klasse von sich dynamisch mit der Zeit verändernden verteilten Systemen.

Diese Arbeit wurde vom Fonds für Wissenschaft und Forschung über das Projekt RiSE (S11405) unterstützt.

Contents

CHAPTER 1

Introduction

The topic of this thesis are "easy[1] impossibility proofs" in the area of distributed computing. Impossibility results in general are an amazing accomplishment, as they allow us to state that any conceivable algorithm will fail to solve a certain problem. Since there are usually innumerably many algorithms that could be devised for a given problem, this means that an impossibility result proves that no algorithm, irrespective of its ingenuity, can possibly succeed.

Probably the most well-known example for an impossibility proof in computer science is the proof of the undecidability of the halting problem by Alan Turing. There it is shown, using a self-reference argument, that there exists no algorithm which can decide for an arbitrary algorithm $\mathcal{A}$ whether or not $\mathcal{A}$ terminates. Whereas this impossibility in a way tells us that we should not even attempt to algorithmically solve the halting problem, the impossibility proofs of this thesis – and in distributed computing in general (c.f. [1]) – are usually of a somewhat different flavor: they provide us with lower bounds, i.e. fundamental limitations that make a problem unsolvable.

From the given assumptions, for which we want to show that solving a problem is impossible, we derive at least one worst-case scenario, where any given algorithm would fail to solve the problem. Nevertheless, if we have a way to determine the assumption coverage of our model, we might find that these cases are rare enough to be tolerable in practice. Alternatively, we might be able to eliminate these cases by strengthening the model, typically via additional engineering efforts, such that the system supplies us with additional guarantees at the expense of some additional cost. Thus, the importance of the impossibility proofs as presented in this thesis stems from isolating and elaborating critical aspects of systems that renders a certain problem unsolvable.

In this work, we study the problem of k-set agreement, which is a natural generalization of the consensus problem. Consensus describes the task where multiple processing entities, called processes, with some means of communication (e.g. the sending/receiving of messages or access to some shared data storage) each start with some initial value and have to agree on one of

[1]In this context, an "easy" proof means one that does not rely on arguments from algebraic topology or point set topology.

the starting values, after a finite period of exchanging information and local computation. The relevance of consensus originates from distributed, i.e., non-centralized, decision making in distributed applications. Instances of consensus prove useful in synchronization, leader election, mutual exclusion and other pivotal services in distributed systems. Moreover, it has been shown that under certain conditions the consensus problem is equivalent to reliable broadcast [31]. While finding algorithms for consensus is easy in systems where the processes as well as communication are reliable, the problem becomes more difficult (some may argue more interesting) in systems prone to faulty behavior. For example, the celebrated FLP result by Fischer, Lynch and Paterson [27] revealed that consensus is impossible in asynchronous systems if just a single process may crash. Subsequent studies of consensus in models with a higher degree of synchrony, such as partially synchronous models or completely synchronous models, revealed a strong dependency between the impossibility of consensus and the synchrony guarantees of the system [22, 25].

k-set agreement generalizes consensus in such a way that, instead of only one, up to k values may be decided on – thus, for $k = 1$, k-set agreement becomes consensus. Besides of its interest for exploring the solvability/impossibility border of distributed computing models, k-set agreement holds significant relevance as a gracefully degrading version of consensus: Assume for example a setting where communication between some clusters is inhibited for a certain period of time, as it could occur e.g. in dynamically deployed wireless systems. If there are k clusters in this system, solving k-set agreement could correspond to solving consensus in each isolated cluster. When the communication between all clusters is enabled again, solving k-set agreement might actually result in solving consensus in the system. Such graceful degradation is usually desired in systems with high availability requirements and/or significant maintenance cost. At the algorithmic level, it requires code that is unaware of k.

The seminal FLP proof for the consensus impossibility in asynchronous systems with crashes utilizes combinatoric arguments [27]. However, most existing corresponding impossibility results for k-set agreement, which show that k-set agreement is unsolvable in asynchronous systems with k crash failures, rely heavily on topological arguments, sharing a connection to Sperner's Lemma [14,19,32]. In order to simplify these proofs and to gain additional insight into the underlying reasons leading to impossibilities, there has been some recent research on showing k-set agreement impossibilities without utilizing topological methods (for instance [4, 5, 8]). This thesis contains a collection of such non-topological proofs: Besides discussing a small selection of interesting existing results in detail, we derive previously unknown results for various models and systems. Note that two papers [12, 46] presenting these results are being submitted to a scientific conference and a journal.

1.1 Motivation

The Information Age. When thinking about the advances that have been made in the area of digital information processing systems during the past decades, the amazing increase of transistors per chip is arguably the main factor driving the increasing miniaturization and pervasiveness of today's computer systems. However, as the processing power of the hardware grew exponentially over the years, the extent in complexity of the software that controls and utilizes these processing units became apparent. Deepening our understanding of computer systems as a whole is the goal of the academic research field of computer science. In order to be able to appropriately describe and advance our understanding in this field, many primarily discrete theoretical and mathematical methods are employed to find suitable models for the questions at hand. This form of mathematical analysis has lead to a multitude of insights and accomplishments that would have hardly been obtainable in established fields of natural sciences such as theoretical physics or theoretical chemistry as well as engineering sciences like theoretical electrical engineering. In computer science, this more abstract form of research has proven to be a pillar on which today's computer systems are built, by sparking technologies such as compilers, operating systems and databases.

Distributed computer systems are one of the key aspects of information technology today. Since the early 1990s, the Internet has increasingly become a driving factor for many facets of our society. In addition, networked embedded systems have emerged to further augment the utility of existing machines and devices and to optimize the efficiency of industrial automation. The past decade has also marked the advent of multicore architectures, the dominant design paradigm of modern CPUs. Among the reasons for this renewed focus on multicore chips are the physical limits imposed by the increasing transistor miniaturization and growing processing speed.

As the demand for these technologies grew, the complexity of developing distributed computer systems as well as algorithms that exploit multicore architectures efficiently, has become apparent. This is especially the case for applications with more stringent requirements such as safety critical real-time systems, where correct design and implementation are of utmost importance. Thus, the need for a sound theoretical study of these systems that serves to deepen our understanding of them grows steadily. The field of distributed computing meets this need as it is the formal and abstract study of distributed computer systems, encompassing the development of models and algorithms for both networked systems and multicore systems.

Impossibility results, in particular, are an integral part of distributed computing research. Apart from their previously mentioned usefulness in identifying and isolating system properties that are crucial for a given problem, they also provide a means to rigorously determine the quality of some given algorithm. If the algorithm succeeds under a system model such that modifying the model incrementally (e.g. adding an additional possible crash failure) results in the impossibility to solve the problem at all, then this algorithm is *optimal* with respect to this system property (e.g. the number of potential crash failures in the system). In this sense, an impossibility result is equivalent to a lower bound for every algorithm.

To some extent, impossibility results even play a role in guiding the research of distributed computing. Since there exist already a multitude of impossibility results for a great variety of problems and system models (see e.g. [7,26,34]), researchers may find extremely useful starting points for further investigation there. Moreover, as mentioned in [26], it often proves useful to develop impossibility results in parallel with an algorithm: If no algorithm can be found for a specific reason, then maybe this very reason contributes towards a general impossibility. Or, vice versa, if no impossibility can be found, the reasons for this could be exploited by an algorithm that actually solves the problem. Additionally, it was the impossibility of certain problems that lead to the active research area of failure detectors (cf. [15, 16]), where, simply put, the goal is to find the properties an oracle must provide in order to make an unsolvable problem solvable, often with the additional constraint that the oracle is weaker than all other oracles that could achieve this. It hence seems particularly promising to use the methods described in this thesis in the still ongoing search for the weakest failure detector for message-passing k-set agreement, as we will elaborate in Chapter 3.

The focus of this thesis are easy (non-topological) impossibility results for k-set agreement. Such non-topological results are important because, as already mentioned, they may supply us with a more immediate understanding of the involved fundamental limitations than their complex topological counterparts. Of course, we do not intend to diminish the relevance and achievement of topological impossibility proofs for k-set agreement, but merely stress the scope and limitations of the non-topological variants. An interesting aspect of the topological proofs is that they essentially show a perpetual preservation of "undecidedness", made possible by a clever modeling of executions of algorithms as structures where certain symmetries can never be broken. We describe in detail a combinatoric interpretation of this idea in Chapter 7. In contrast, the proofs that we developed in the course of this thesis are essentially based on a partitioning argument, where the core is always an isolation of components of the system that inevitably leads to the algorithm making a mistake. The question whether and how these different causes for impossibilities are connected was one of the key issues that motivated this work.

1.2 Short Overview of Related Work

In their breakthrough result, Fischer, Lynch and Patterson showed that consensus is impossible in asynchronous message passing systems even with just a single crash failure [27]. This result may be regarded as the dawn of impossibility proofs in the field of distributed computing [26]. It is established employing a so-called bivalence proof, a concept which we will introduce in more detail in Chapter 2.

Dolev, Dwork and Stockmeyer extended this result, by providing a comprehensive set of proofs for varying system parameters that influence the solvability of consensus in [22]. They investigated the fundamentals of impossibility proofs for consensus and found very general criteria that imply impossibility in message passing systems. They provide generic theorems and lemmas that are then used to show concrete impossibilities, respectively possibilities, for consensus for a total of 32 different combinations of model parameters. Their research resembles the key aspect that forms the motivation behind many impossibility proofs in this field, i.e., identifying and isolating those aspects of a system model that determine the solvability of a given problem, as mentioned previously in this chapter.

In their seminal work, Santoro and Widmayer presented a model for dynamic communication faults and showed the fundamental impossibilities for consensus in such settings [43]. They divide possible communication faults into three categories and present lower bounds and impossibilities for consensus for each of them. A comprehensive overview of the topic of dynamic link failures, which also includes new results that have been applied in this thesis, has been published by Schmid, Weiß and Keidar in [44].

Encouraged by the unsolvability of consensus, researchers began to focus on the natural generalization of consensus, k-set agreement. Borowsky and Gafni, Chaudhuri, and Herlihy and

Shavit (independently) proved the impossibility of k-resilient (i.e. tolerating up to k crashes) k-set agreement in [14, 19, 32]. Although their approaches differ significantly in detail, the idea common to all of them is modeling some crucial properties of certain executions of algorithms as topological simplices, to which Sperner's Lemma may be applied. This essentially enables the infinite preservation of a non-terminal state of a sufficiently large "core" of processes that implies the desired impossibility. As these results require arguments from algebraic topology, we will not go into their details in this thesis, but instead focus on recent corresponding non-topological impossibility results for k-set agreement: Some non-topological impossibilities for k-set agreement have recently been published by Attiya and Castañeda in shared memory [4] and by Attiya and Paz in [5]. We will discuss these results and their connection to different non-topological approaches for the impossibility of k-set agreement in detail in Chapter 7.

The impossibility proof for k-set agreement motivated the search for the weakest failure detector that makes k-set agreement solvable. While the weakest failure detector for shared memory systems has already been found [20, 21], the search for the weakest failure detector for k-set agreement in message passing systems is still ongoing. A comprehensive overview of this research can be found in [13]. We will discuss some important results related to this topic in Section 3.5 and explain why non-topological impossibility results may be a promising tool for closing this gap in our knowledge about distributed computing.

Biely, Robinson and Schmid have recently presented a generic theorem for showing impossibilities of k-set agreement in message passing systems [8]. This theorem is applicable to a variety of system parameters and will be referenced and applied throughout this thesis. We provide an overview and a detailed description of the theorem in Chapter 3. A comprehensive survey on the topic of impossibility results in distributed computing has been elaborated by Fich and Ruppert [26]. There, they present many of the fundamental principles that are employed in impossibility proofs, discuss and rigorously compare various models frequently used in distributed computing and finally present many existing impossibility results.

Regarding algorithms for consensus and k-set agreement in dynamic networks, we refer to the publications [9, 10] by Biely, Robinson and Schmid: Key aspects of these models as well as selected theorems on which we will rely upon in order to derive new results will be elaborated in Chapters 4 and 5.

Raynal and Travers published a k-set agreement algorithm that is optimal with respect to the number of tolerated general omission failures in [41]. General omission failures are a more general type of failures than crash failures and include crash failures as a special case. We will provide a short presentation of this algorithm in Chapter 6.

Finally, the books of Attiya and Welch [7] and Lynch [34] provide an extensive introduction into the field of distributed computing as well as a thorough presentation of many advanced topics along with pointers to further literature.

1.3 Outline and Major Contributions

- Chapter 2 provides a formal introduction of our generic computing model. We state the common properties and provide an overview of all the computing models used in this thesis. Some details, such as the precise failure assumptions, will be introduced in the according chapters.

- Chapter 3 repeats and analyzes a generic impossibility theorem for k-set agreement. Subsequently, we will often reference and apply this theorem in various models and scenarios. A selection of these applications will be published in a journal version of the original paper from [8] that is about to be completed.

- Chapter 4 introduces systems where communication is highly dynamic, as it occurs e.g. in networked portable devices or wireless sensor networks. We model such systems, using sequences of directed graphs that determine the actual communication among processes over time. We call these systems synchronous directed dynamic networks (DDNs) and establish the following new impossibilities for DDNs:

  – Impossibility of consensus with moving root components
  – Impossibility of k-set agreement with a single moving root component
  – Impossibility of k-set agreement where a single root is stable only for short intervals
  – Impossibility of k-set agreement where decisions can be hidden from future root components

  We used the latter impossibility in a related publication [46], which is under review at the time of writing, to find a weak assumption that enables solving k-set agreement in this setting and an according algorithm.

- Chapter 5 explores the foundations of DDNs starting out from a compact predicate $\mathcal{P}_{\text{srcs}}(k)$ that is sufficient to solve k-set agreement. We provide the following new impossibility results:

  – A novel predicate $\mathcal{P}^r_{\text{srcs}}(k)$, which is slightly weaker than $\mathcal{P}_{\text{srcs}}(k)$, makes k-set agreement impossible.

- Chapter 6 analyzes the scenario where senders and/or receivers may be faulty, i.e., in contrast to DDNs, communication failures are modeled by means of process faults. We provide the following new non-topological prof for a known lower bound on the time complexity:

  – In systems where t processes may exhibit general omission faults, k-set agreement is impossible in less than $\frac{t}{k}$ rounds.

- Chapter 7 explores the setting where communication occurs via shared memory, a model that provides much stronger guarantees to the algorithm than the message-passing models discussed in the previous chapters and therefore makes showing impossibilities more difficult.

  – We investigate what parts of the BRS-Theorem are applicable in the shared memory model and what remaining part is not applicable and why.

CHAPTER 2

Model of Computation

2.1 A Distributed State Machine

We model a distributed computing system as an ensemble or set of processes Π. Each process $p_i \in \Pi$ knows its unique identifier i that ranges between 1 and $n = |\Pi|$, the total number of processes in the system. A single process, executing an instance of a deterministic algorithm is modeled as a deterministic state machine. It possesses an unlimited amount of memory that comprises its current state as well as some means of exchanging data with other processes, as described in Section 2.2. Each process is associated with a transition function that maps to each state a single successor state. We usually specify the transition function itself using an algorithm in pseudo-code. We say a process takes a step when a process performs the transition from a state to its successor state. In our model steps are instantaneous and atomic; however, we might impose certain restrictions on the frequency with which processes may take steps relative to each other through synchrony assumptions, as described in Section 2.3.

We call the set of states of all the processes in the system a *configuration* of the system, denoted C_i. C_0 denotes an initial configuration, composed of initial states of all processes. A transition from a configuration C_{k-1} to a successor configuration C_k is triggered by an event ϕ^k, which is said to occur *at time* k. This concept of time describes a global time that is usually not observable by a process but used for analysis purposes only.

An event ϕ^k may be a computation step of an individual process or a communication event, for example the reception of a message, as described in the next section. Sometimes it is useful to group single events together into compound events. It could for example be beneficial to abstract from single computation events in systems where processes are controlled by a central clock and instead consider a single computation event at every process as one compound event. In the context of impossibility proofs we usually think of the events being scheduled by an *adversary* that aims to derive a special case where any algorithm must fail. We usually aim to restrict the capabilities of the adversaries in a sensible way, as described in Section 2.3.

We call a (possibly infinite) sequence of configurations and events $C_0, \phi_1, C_1, \phi_2, \ldots$ an *execution* or *run* of an algorithm. In the exemplary configuration tree of Figure 2.1, an execution

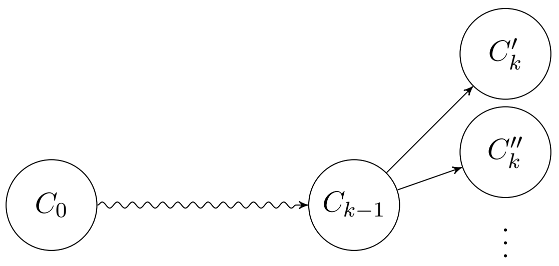


Figure 2.1: Configuration C_{k-1} is reachable from the initial configuration C_0, as indicated by the wavy line. This means that there is a sequence of configurations that starts in C_0 and ends in C_{k-1}, where each configuration (except for C_0) in this sequence is a successor of the previous configuration. Configurations C'_k, C''_k and possibly many more (as indicated by the three dots) are successor configurations of C_{k-1}.

corresponds to a single path through the tree.

An execution is called *admissible* if it adheres to the fairness and failure assumptions of the model. For instance, this usually requires that in an infinite execution every process takes an infinite number of steps, which essentially corresponds to the constraint that no process may starve. An algorithm is said to be correct or solve a given problem, if it generates only executions that adhere to the problem specification. We usually state the problem specification itself using safety and liveness conditions. Intuitively, a *safety conditions* corresponds to the requirement that "something bad will never happen", while a *liveness condition* asserts that "something good will eventually happen". Formally, in a correct algorithm, a safety condition must be met in every prefix of an execution (e.g. a process may never stop to make any steps), while a liveness condition must be met in every execution some number of times, possibly even an infinite number of times (e.g. a process must make an infinite number of steps).

Two executions of an algorithm α, β are *indistinguishable* for a a process p, denoted $\alpha \overset{p}{\sim} \beta$ if p observes no difference between α and β, i.e. if p's view of the distributed system is the same in α and in β. In particular, p executes the same state transitions in α and β. For example in the case of a system where communication occurs via the sending and receiving of messages, this corresponds to p starting in the same state and receiving the same messages in α and β. If α and β are indistinguishable for all processes in a set of processes D, we write $\alpha \overset{D}{\sim} \beta$.

Indistinguishability of executions is a central concept in impossibility proofs: Proving that there does not exist an algorithm that solves a certain problem is possible by showing that any algorithm that would solve this problem also generates executions which are indistinguishable for some process from executions where the problem remains unsolved.

2.2 Communication

We will study two different primitives for communication among the processors of a distributed system. The first is *message passing*, where processes communicate by the sending and receiving of messages. The second is *shared memory* where all processes have access to a common data storage.

Asynchronous message passing can always be simulated in asynchronous shared memory, however the other direction of the simulation is not always possible (this possibility depends on the number of allowed crash failures) [7, Section 5.33 and Theorem 10.22]. This leads to the conclusion that the shared memory model possesses more expressive power; from the perspective of the algorithm, shared memory allows event ordering by means of "write before read", for example, which cannot be implemented in message passing. Therefore, message passing is a less powerful yet more fundamental model than shared memory.

Message Passing

We use a model based on [22] and describe message passing by using a local buffer at every process that contains the messages which have been sent to that process but not yet delivered. In their transition function, i.e. during a step, processes may access a send(p, m) function that puts the message m into p's buffer. We call this function the message sending function. This enhances the set of events in our model by a send event and extends the state transition function of a process to depend not only on the local state but also on the contents of its message buffer.

The time it may take from calling send(p, m) until m is actually put into p's buffer depends on our synchrony and failure assumptions, as described in the next section.

Shared Memory

A shared memory variable X is denoted by an uppercase letter and can be accessed by any process through instantaneous (zero-time) read(X) and write(X) operations. Consequently, the state transitions in a shared memory system depend not only on the local state of the processes but also on the contents of the shared memory.

In this thesis we will focus on single-writer/multi-reader (1WnR) registers. These shared memory variables can be written only by a single dedicated process, while every process can read their content. The more fundamental single-writer/single-reader (1W1R) registers can be used to construct single-writer/multi-reader registers at the cost of some efficiency [40].

We will furthermore restrict ourselves to *immediate snapshot (IS) executions*. An IS execution $\alpha = s_1, s_2, \ldots$ consists of a sequence of non-empty sets s_i of processes making steps, where shared memory access of the processes in s_i is restricted as follows: One process after the other writes to the shared memory in some arbitrary sequence, followed by all processes of s_i reading from the shared memory simultaneously. We denote a r-times repetition of s_i by s_i^r.

Only considering IS executions may seem, at first glance, like an undue simplification of the model that saves us the trouble of considering more complex cases. In truth, however, when we are concerned with showing impossibilities, using a stronger model that supplies additional guarantees to the algorithm only strengthens the impossibility result: If the stronger case already leads to an impossibility, i.e., to an algorithm-independent execution that does not solve the problem, then this very same execution can also be obtained when running the algorithm in the more general version of the model, as long as the admissible executions in the relaxed variant are a superset of the restricted version. This is an important aspect of many impossibility proofs [26].

2.3 Synchrony and Failures

A commonly used perspective is to view the execution of a distributed algorithm as a game versus an adversary. The adversary is in control of certain important parts of the system, that are outside the sphere of control of the algorithm, in particular occurrence times of computing events and failures, according to some rules set forth in the system and failure model. As developers we need to design an algorithm that implements a winning strategy, i.e., wins no matter what the adversary does within its model.
The power of the adversary is restricted by our model, in particular w.r.t. the scheduling of the processes, the delay of the messages and the type and number of failures.

Synchrony defines the timing properties of the system, thereby establishing the rules for the scheduling of the processes' computation as well as the delay between sending and receiving a message by the adversary. Failure assumptions limit the amount and type of failures that may occur in our system. There are many different synchrony and failure assumptions that prove useful in a multitude of applications, however we will illustrate only a small selection that is relevant in our context here.

This means that, given a basic model for a distributed system, the synchrony and failure assumptions are the means by which we can strengthen or weaken the adversary.

Synchrony of Processes

We already stated that the steps a process makes are atomic and instantaneous in our model. In real-world computer systems, however, executing instructions incurs a delay, which is the source of many interesting questions in computer science. We take this into account in our model by allowing finite time to pass between steps. We call a sequence of steps (determined by the delays) a *schedule* of the system. When proving the correctness of an algorithm, this schedule is chosen by the adversary from the set of allowed schedules as defined by the process synchrony assumptions. We give two examples for such synchrony assumptions:

Asynchronous processes is equivalent to the absence of any restriction on the process scheduling, i.e. any process can take a step, according to its transition function, at any time and any finite delay may occur between these steps. This corresponds to the most powerful adversary regarding the process scheduling.

With the assumption of *synchronous processes*, a schedule consists of a sequence of time intervals of identical length, where all processes simultaneously execute one step. We call one such interval a *round* of a synchronous schedule. Synchronous processes are a stronger assumption than asynchronous processes.

Synchrony of Messages

We consider synchronous messages only in conjunction with synchronous processes, where computation proceeds in lock-step rounds. Each round consists of the sending of messages, the delivery of all pending messages, and a computation step at every process.

Asynchronous messages means that the message delay may be arbitrarily large but finite. This implies that in an admissible execution, messages never get lost and must eventually arrive, yet their delay may be unbounded. This is particularly important in a scenario where a process might crash and messages are asynchronous: It may be impossible for a process that waits for an incoming message to determine whether this message will still arrive at some point in the future or if the process that was supposed to send the message has crashed.

Failures

Real computer systems are prone to errors which in turn may lead to failures. Modeling failures of computing systems is a concern in many areas of computer science, especially in dependable systems engineering. The range of failure models is considerable: Byzantine failures [33] allow processes to exhibit arbitrary behavior that is completely independent of the transition function. This means that Byzantine failures enable us also to model an intelligent attacker, aiming at compromising the system. In the main part of this thesis, however, we will restrict ourselves to crash failures, which enable us to model a process that ceases operating. We will introduce more severe failure models later in Chapter 6.

A *crash failure* is said to occur at a processes if, from a certain instant on, the process stops to make any steps. This instant is completely arbitrary, which means that the process may even crash during a step. An especially insidious kind of crash is one where a process crashes while sending a message: in this case the message may reach only a subset of the intended recipients. A priori we do not know which processes will crash; the failure assumptions only state an upper bound on the number of processes that may crash. Let F to denote the set of faulty processes and let $|F| = f$. An algorithm that tolerates up to f failures is called *f-resilient*.

Following our definition of time as the number of events that happened so far (cf. Section 2.1), we call the set of processes that crashed in an execution by time t the failure pattern $F(t)$. The set of allowed failure patterns is called the environment $\mathcal{E}$ of our system. For instance, considering only up to f crash failures corresponds to setting the environment to include only those failure patterns where $|F(t)| \leq f$ for all $t \geq 0$. Another popular example is the *wait-free environment* where all processes but one may crash.

2.4 Consensus and k-set Agreement

Consensus and, more generally, k-set agreement are problems where the processes hold an initial proposal value x_i, which is taken from a finite set V. In the case of binary consensus $|V| = 2$ and usually $V = \{0, 1\}$. For k-set agreement, we assume $|V| > k$ to circumvent trivial solutions. The processes must agree upon a single non-trivial final value that is assigned once and irreversibly to a write-out output variable / value y_i during an execution. We call y_i the decision value. An algorithm that solves consensus must satisfy the following properties:

Agreement No two correct processors may decide on a different y_i.

Validity If $y_i = v$ then v is some p_i's x_i.

Termination Eventually every correct p_i assigns a value to y_i.

While the agreement and termination conditions are rather straight-forward, validity ensures that there can be no trivial decision value, thereby excluding algorithms where the processes decide on a predetermined value. Termination is a liveness property, whereas agreement and validity are safety properties.

Modifying the agreement condition by demanding that all processes – in particular, even the faulty ones that decide – must agree on the same y_i, results in the stronger *uniform agreement* condition and the resulting *uniform consensus* problem. While in synchronous systems uniform consensus is harder to solve than consensus, for purely asynchronous systems, the two problems are equivalent [17]. The reason for this is that, in an algorithm that solves consensus in an asynchronous system, a process that decides must be in agreement with all other processes, in particular also with those that are very slow. Since a deciding process is unable to distinguish between very slow processes and crashed processes, uniform agreement is hence mandatory.

k-set agreement is a generalization of consensus where, instead of a unique decision value, k different decision values are permitted system-wide [19]. In this sense, consensus is the special case of k-set agreement where $k = 1$. k-set agreement translates to replacing the agreement property of consensus by the following k-agreement property:

k-Agreement There may be no more than k different decision values y_i obtained by correct processes.

In their seminal work, Fischer, Lynch and Paterson managed to prove that consensus is impossible in a completely asynchronous system with even a single crash failure, using only combinatoric arguments [27]. Subsequently, researchers tried to find a similar bound for k-set agreement, which turned out to require quite different methods.

An impossibility result for k-set agreement, which based on topological arguments, was found independently from each other by three different research groups. They showed that k-set agreement is impossible if k or more crash failures can occur [14, 32, 42]. All these results share a connection to Sperner's Lemma which enables the perpetual preservation of an undecided configuration. We will present a non-topological variant of this idea later in Chapter 7.

Note that, for $k = 1$, this result matches exactly the consensus impossibility. Observe that, for less than k failures, a simple algorithm exists in asynchronous systems: k predetermined processes propose a value by communicating it to the processes in the distributed system and every process decides on the first value it receives. We will usually assume that $k < n$, as k-set agreement can be trivially solved when $k = n$ by an algorithm where every process decides on its input value.

Bivalent and Univalent Configurations

An important tool for proving impossibilities for binary consensus ($|V| = 2$) was introduced along with the original proof for the asynchronous consensus impossibility with crash failures [27]: the valence of a configuration. Proofs that use the valence of a configuration are commonly referred to as "bivalence proofs". By definition every execution of a binary consensus algorithm must lead to exactly one of the two possible decision values 0 and 1. For the value $v \in \{0, 1\}$, valences are defined as follows:

- A configuration C is called *v-decided* if all processes have decided on the value v.
- A configuration C is called *v-valent*, if a v-decided configuration is reachable from C and no $(1 - v)$-decided configuration is reachable from C. A v-valent configuration is sometimes also called *univalent*.
- A configuration C is called *bivalent*, if there is a v-valent and a $(1-v)$-valent configuration reachable from C. Note that a bivalent configuration cannot be a terminal configuration for consensus, i.e., a configuration where every process has decided. In fact, no process can have decided in a bivalent configuration.

A bivalence impossibility proof is usually a proof by induction: It first establishes that there is a bivalent initial configuration and then proceeds to showing that any bivalent configuration has a bivalent successor configuration. The implication of such a bivalence proof is that there is an execution where the configurations are perpetually bivalent – i.e. an execution that never terminates. Thus, the algorithm does not solve consensus in all executions. We usually go even one step further and show that any algorithm has a perpetually bivalent execution in a certain system – thus proving that consensus is impossible in the given system.

2.5 Failure Detectors

Given an impossibility result, which states that there is no algorithm which solves some problem in a certain system, one is often interested in the question what properties of the model made the problem unsolvable. This line of research is especially interesting from an engineering point of view, as it may be possible to adjust a real-world system with reasonable effort in such a way that it corresponds, with a sufficiently high assumption coverage, to a model in which the problem is indeed solvable. A very convenient tool for finding such “solvability boundaries” are failure detectors.

A failure detector is an oracle that can be queried by a process during a computation step (cf. [16]). As their name implies, failure detectors provide a process with information that could usually not be obtained by “regular” means, for example, whether some other process has crashed in a system with asynchronous messages. Just as the Oracle of Delphi however, failure detectors usually don’t give trustworthy answers when queried. Rather, their responses merely adhere to a set of rules, called the failure detector specification, which is characteristic for each type of failure detector.

An important result about failure detectors is that we can often compare them based on the rules they abide by, thereby establishing a partial order of stronger and weaker failure detectors. Using such orderings, we can answer the question of which property of a model is necessary to render a problem solvable, by finding a weakest failure detector for this problem. Interestingly, the weakest failure detector for k-resilient k-set agreement in message passing systems is still unknown, despite 15 years of failure-detector-related research.

Formally, a failure detector maps a process identifier and a point in time to an element from the failure detector’s output range. We call this mapping the *failure detector history* $\mathcal{H}(p, t)$. We define executions where a failure detector may be queried as follows: In an admissible execution, at any point in time t, the failure detector must output only legal histories. A failure detector history is called legal when it is in accordance with the failure detector specification applied to the failure pattern $F(t)$ of the execution, i.e. the set of processes that failed until t.

Comparing Failure Detectors

We already stated that an order among failure detectors may be established and go now into the details of how this can be achieved. The basic idea is that we search for (purely asynchronous) algorithms that transform a failure detector $\mathcal{D}$ to another failure detector $\mathcal{D}'$. An algorithm in a system that is augmented by $\mathcal{D}$ is said to transform $\mathcal{D}$ to $\mathcal{D}'$, in symbols $\mathcal{A}_{\mathcal{D}\to\mathcal{D}'}$, if the processes hold a variable which emulates a failure detector history of $\mathcal{D}'$. The existence of such a transformation algorithm $\mathcal{A}_{\mathcal{D}\to\mathcal{D}'}$ implies that the output of $\mathcal{D}'$ may be deduced from the output of $\mathcal{D}$. Thus, if $\mathcal{A}_{\mathcal{D}\to\mathcal{D}'}$ exists, we call $\mathcal{D}'$ weaker than $\mathcal{D}$ ($\mathcal{D}' \leq \mathcal{D}$), respectively $\mathcal{D}$ stronger than $\mathcal{D}'$. If additionally also an algorithm $\mathcal{A}_{\mathcal{D}'\to\mathcal{D}}$ exists, we call $\mathcal{D}$ equivalent to $\mathcal{D}'$, and if $\mathcal{A}_{\mathcal{D}'\to\mathcal{D}}$ does not exist, we call $\mathcal{D}'$ strictly weaker than $\mathcal{D}$, respectively $\mathcal{D}$ strictly stronger than $\mathcal{D}'$.

From the above definition of an ordering of failure detectors we can conclude that if a problem is unsolvable using $\mathcal{D}$ and $\mathcal{D}' \leq \mathcal{D}$, then the problem is also unsolvable using $\mathcal{D}'$. The reason why this is the case is because, by definition, $\mathcal{D}' \leq \mathcal{D}$ implies that $\mathcal{D}'$ can be "extracted" from some algorithm that uses $\mathcal{D}$. Therefore, combining the solution using D' with $\mathcal{A}_{\mathcal{D}\to\mathcal{D}'}$ would provide a solution using $\mathcal{D}$, which does not exist.

The Failure Detector Σ

The first failure detector we will introduce is the *quorum failure detector* Σ. According to [7, Theorem 10.22] shared memory (or more specifically, atomic registers) may be simulated in message passing systems only if a majority of the processes is correct. It has been shown that Σ is the weakest failure detector to perform this simulation in systems with arbitrary failure patterns, i.e. in any environment $\mathcal{E}$ [29].

Considering that Σ is powerful enough to perform the above simulation, it is perhaps somewhat surprising that Σ merely guarantees to *eventually* output only non-faulty processes. Still, the output of Σ at two different processes is guaranteed at any time to have at least one common element. Formally, Σ satisfies the following two properties for all environments $\mathcal{E}$ and all possible failure patterns $F(.) \in \mathcal{E}$.

Intersection: For any two processes p, p' and any time instants t, t' it holds that $\mathcal{H}(p,t) \cap \mathcal{H}(p',t') \neq \emptyset$.

Liveness: $\exists t \forall t' \geq t \; \forall p \notin F : \mathcal{H}(p,t') \subseteq (\Pi \setminus F)$

In order to satisfy the intersection property for all $\mathcal{E}$, we define the failure detector's history for processes $p \in F(t)$ that are faulty from time t on as $\forall t' \geq t : \mathcal{H}(p,t') = \Pi$.

The Failure Detector Σ_k

The *generalized quorum failure detector* is denoted Σ_k, where Σ corresponds to Σ_1. It was shown that Σ_k must be implementable from any failure-detector based solution for k-set agreement in message passing systems [13]. Like Σ, Σ_k also satisfies an intersection property, albeit for sets of $(k+1)$ processes, thereby matching the intersections for pairs of processes in the case of $k = 1$. The liveness property of Σ_k is the same as in Σ:

Intersection: For all sets $\{p_1, \ldots, p_{k+1}\}$ of $k+1$ processes and all multisets $t_1, \ldots, t_{k+1}$ of $k+1$ time instants, there exist indices i, j with $1 \leq i \neq j \leq k+1$ such that $\mathcal{H}(p_i, t_i) \cap \mathcal{H}(p_j, t_j) \neq \emptyset$.

Liveness: $\exists t \forall t' \geq t \; \forall p \notin F : \mathcal{H}(p, t') \subseteq (\Pi \setminus F)$

As with Σ, we assert $p \in F(t) \Rightarrow \forall t' \geq t : \mathcal{H}(p, t') = \Pi$.

The Failure Detector Ω

Ω stands for the *eventual leader failure detector*. It was shown to be the weakest failure detector that allows to solve consensus in asynchronous message passing systems with a majority of correct processes [15]. The combination $\Sigma \times \Omega = (\Sigma, \Omega)$, whose history is a pair of values, one for Σ and one for Ω, has been shown to be the weakest failure detector for wait-free consensus, i.e. $(n-1)$-resilient consensus [29].

Ω outputs only a single process identifier and eventually will output the same correct (trusted) processes at all non-faulty processes, which can be expressed formally as the following two properties:

Validity For all processes p and all times t, $\mathcal{H}(p, t) \in \Pi$.

Eventual Leadership There exists a time t_{GST} and a correct process q such that for all correct processes p and times $t' \geq t_{\text{GST}}$, $\mathcal{H}(p, t') = q$.

The Failure Detector Ω_k

Ω_k is the *generalized leader oracle* where Ω corresponds to Ω_1. The generalization is that Ω_k outputs a set of k processes, which eventually contains at least one correct process and eventually does not change anymore. Formally, Ω_k is defined as satisfying the following properties for all environments $\mathcal{E}$ and all possible failure patterns $F(.) \in \mathcal{E}$:

Validity For all processes p and all times t, $\mathcal{H}(p, t)$ is a set of k processes.

Eventual Leadership There exists a time t_{GST} and a set of processes *LD* with $(LD \cap (\Pi \setminus F)) \neq \emptyset$ such that for all correct processes p and times $t' \geq t_{\text{GST}}$ it holds that $\mathcal{H}(p, t') = LD$.

Ω_k alone already allows solving k-set agreement in message passing systems with a majority of correct process [35]. The combination of $\Sigma_k \times \Omega_k = (\Sigma_k, \Omega_k)$ outputs a value for Σ_k and one for Ω_k. $(\Sigma_{n-1}, \Omega_{n-1})$ is able to solve $(n-1)$-set for agreement [13], however, it is shown in [8] that (Σ_k, Ω_k) is not strong enough to solve wait-free k-set agreement for all k. This result will be investigated in detail in Section 3.4.

CHAPTER 3

A Generic Impossibility Theorem

This chapter contains a detailed description of the non-topological impossibility theorem, originally published as [8, Theorem 1] and subsequently called BRS-Theorem, with additional discussions and explanations.

3.1 Intuition

In order to prove that k-set agreement is impossible in the model of a distributed computing system with a given communication primitive and given synchrony and failure assumptions, it suffices to show that any k-set agreement algorithm has at least one run where more than k decision values are decided upon. In other words, if we could somehow exploit the inherent system properties in such a way that, independent of a concrete algorithm, there are too many decision values, we had shown impossibility of k-set agreement in the given model.

This idea is utilized in the BRS-theorem: In order to apply the theorem to a given system model, we need to find a partitioning, i.e., pairwise disjoint, non-empty sets of processes $D_1, D_2, \ldots D_k$ with $\bigcup_{i=1}^{k} D_i = \Pi$. To stress its special role, we write $\overline{D}$ for the set D_k. We choose the partitions such that consensus is impossible in $\overline{D}$ (by this we mean that the processes of $\overline{D}$ could not solve consensus when left to run isolated from the rest of the system), there are runs where processes start with distinct proposal values, and the following properties are satisfied (cf. Figure 3.1):

(dec-D) For every set D_i, $0 < i < k$, a distinct value v_i was proposed by some $p \in \bigcup_{i=1}^{k-1} D_i$, and there is some $q \in D_i$ that decides v_i.

(dec-$\overline{\mathbf{D}}$) The processes in $\overline{D}$ may not receive any messages from a process $p \in \Pi \setminus \overline{D}$ until every process in $\overline{D}$ has decided.

Note carefully that (dec-$\overline{D}$) implies that $\overline{D}$ is partitioned from the rest of the system (also) in terms of communication: No process of $\overline{D}$ may hear from any process of $\Pi \setminus \overline{D}$ before making a decision. Actually, although not explicitly required by (dec-D) and (dec-$\overline{D}$), it is typically also necessary that every other D_i is partitioned from the rest of the system as well in order to guarantee the distinct decision v_i.

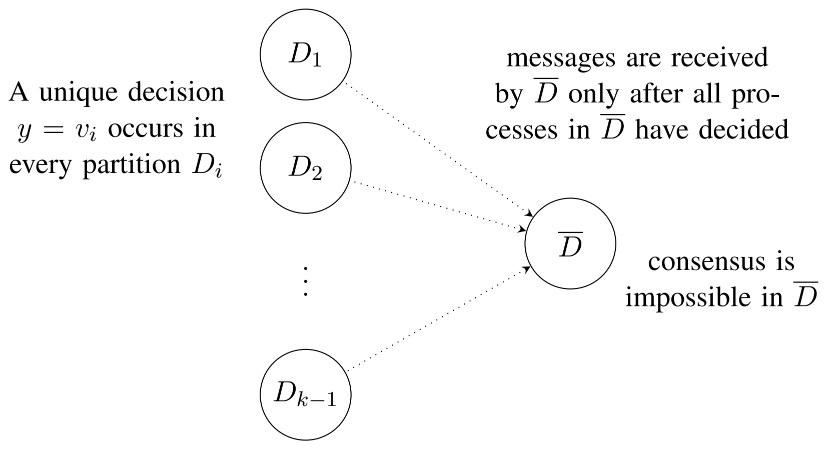

A unique decision $y = v_i$ occurs in every partition D_i
D_1
D_2
D_{k-1}
messages are received by $\overline{D}$ only after all processes in $\overline{D}$ have decided
$\overline{D}$
consensus is impossible in $\overline{D}$

Figure 3.1: A k-partitioning

If such a partitioning exists, there is an intuitive explanation why k-set agreement is impossible: Consider a run of an Algorithm in a system where such a partitioning can be applied and every process starts with a unique initial value. Property (dec-D) provides us with the processes in $\Pi \setminus \overline{D}$ having at least $k-1$ distinct decision values that are different from the initial values in $\overline{D}$. By property (dec-$\overline{D}$), $\overline{D}$ may receive information from any other partition only after all processes in $\overline{D}$ have decided. Therefore, in order to be able to establish the safety property *k-agreement* in the system, the processes of $\overline{D}$ must decide on the very same value, which contradicts that consensus is impossible in $\overline{D}$.

3.2 The BRS-Theorem

The theorem uses a simulation that allows us to express running an algorithm designed for the n processes Π only on a subset of processes $D \subseteq \Pi$. This is done by modifying the message sending function such that, when a process initiates a broadcast, the message is only sent to the processes in D. We call this simulation a restriction of an algorithm and define it formally as follows:

Definition 1 (Restriction of an Algorithm from [8, Definition 1])**.** *Let $\mathcal{A}$ be an algorithm that works in system $\mathcal{M} = \langle\Pi\rangle$ and let $D \subseteq \Pi$ be a nonempty set of processes. Consider a restricted system $\mathcal{M}' = \langle D\rangle$. The **restricted algorithm** $\mathcal{A}_{|D}$ for system $\mathcal{M}'$ is constructed by dropping all messages sent to processes outside D in the message sending function of $\mathcal{A}$.*

Note that, according to this definition, for the processes running $\mathcal{A}_{|D}$ it looks like they are part of a larger system $\langle\Pi\rangle$ and just don't receive any messages from any process outside of D. In particular, they perceive the number of processes in the system as $|\Pi|$. Nevertheless, $\mathcal{A}_{|D}$

is a valid distributed algorithm for a distributed system made up of the processes in D only. Obviously, it would also be possible to drop any dead code from the transition relation of $\mathcal{A}_{|D}$ (devoted to processing messages from $\Pi \setminus D$, for example).

The next definition is required to compare two sets of runs with regard to their indistinguishability for a fixed set of processes D. Note that the indistinguishability $\alpha \overset{D}{\sim} \beta$ here means that α and β are indistinguishable for the processes in D until decision, i.e. it is slightly different from the notion introduced in Section 2.1, which holds throughout the entire execution.

Definition 2. *[Compatibility of Runs from [8, Definition 3]] Let $\mathcal{R}$ and $\mathcal{R}'$ be sets of runs. We say that* ***runs $\mathcal{R}'$ are compatible with runs $\mathcal{R}$ for processes in* D**, *denoted by $\mathcal{R}' \preccurlyeq_D \mathcal{R}$, if $\forall \alpha \in \mathcal{R}' \, \exists \beta \in \mathcal{R} \colon \alpha \overset{D}{\sim} \beta$.*

Finally, let $\mathcal{M}_\mathcal{A}$ denote all runs of algorithm $\mathcal{A}$ in system $\mathcal{M} = \langle \Pi \rangle$, and $\mathcal{M}'_{A_{|\overline{D}}}$ be all runs of $\mathcal{A}_{|\overline{D}}$ in system $\mathcal{M}' = \langle \overline{D} \rangle$. We can now repeat the BRS-Theorem:

Theorem 1 (k-Set Agreement Impossibility from [8, Theorem 1])**.** *Let $\mathcal{M} = \langle \Pi \rangle$ be a system model and consider the runs $\mathcal{M}_\mathcal{A}$ that are generated by some fixed algorithm $\mathcal{A}$ in $\mathcal{M}$, where every process starts with a distinct input value. Fix some nonempty and pairwise disjoint sets of processes $D_1, \ldots, D_{k-1}$ and a set of distinct decision values $\{v_1, \ldots, v_{k-1}\}$. Moreover let $D = \bigcup_{1 \le i < k} D_i$ and $\overline{D} = \Pi \setminus D$. Consider the following two properties:*

(dec-D) *For every set D_i, $0 < i < k$, value v_i was proposed by some $p \in \bigcup_{i=1}^{k-1} D_i$, and there is some $q \in D_i$ that decides v_i.*

(dec-$\overline{\text{D}}$) *If $p_j \in \overline{D}$ then p_j receives no messages from any process in D until every process in $\overline{D}$ has decided.*

Let $\mathcal{R}_{(\overline{D})} \subseteq \mathcal{M}_\mathcal{A}$ and $\mathcal{R}_{(D,\overline{D})} \subseteq \mathcal{M}_\mathcal{A}$ be the sets of runs of $\mathcal{A}$ where (dec-$\overline{D}$) respectively both, (dec-D) and (dec-$\overline{D}$), hold. Suppose that the following conditions are satisfied:

(A) *$\mathcal{R}_{(\overline{D})}$ is nonempty.*

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$

In addition, consider a restricted model $\mathcal{M}' = \langle \overline{D} \rangle$ such that the following hold:

(C) *Consensus is impossible in $\mathcal{M}' = \langle \overline{D} \rangle$*

(D) $\mathcal{M}'_{A_{\overline{D}}} \preccurlyeq_{\overline{D}} \mathcal{M}_A$

Then, $\mathcal{A}$ does not solve k-set agreement in $\mathcal{M}$.

Note that the theorem requires four properties of a partitioning that make k-set agreement impossible. This means that if some algorithm $\mathcal{A}$ suffers from such a partitioning, the theorem implies that $\mathcal{A}$ cannot solve k-set agreement. When proving the general impossibility of k-set agreement for a given system, applying the theorem will usually entail showing that all possible algorithms designed for this system allow a partitioning that satisfies these properties.

We will now discuss why this formalization is equivalent to the intuition presented in the previous section. The property that at least one run of $\mathcal{A}$ satisfies (dec-D) and (dec-$\overline{D}$) is expressed in (A) and (B). (A) states that there exist runs where property (dec-D) holds. (B) states that for every run ρ where property (dec-D) holds, there exists a run ρ', indistinguishable to the processes in $\overline{D}$, where both properties (dec-D) and (dec-$\overline{D}$) hold. Under the assumption that both (A) and (B) hold, we have that, because ρ exists, so does ρ' and thus there exists a run ρ' that satisfies (dec-D) and (dec-$\overline{D}$).

The reason why the theorem does not directly state that there are runs where both properties hold has to do with what the impossibility of solving a problem implies: Recall from the previous section that we will use the impossibility of consensus in $\overline{D}$ to derive the impossibility of k-set agreement in the whole system. Moreover, from the impossibility of consensus we may only conclude that there exists an execution for $\overline{D}$ where consensus is not solved. Informally, in order to "translate" this to an impossibility result for k-set agreement, we need that this very execution of $\overline{D}$ happens in a run where we also have $k-1$ unique decisions in D. Obviously, if we did not enforce this somehow, the execution(s) where consensus fails in $\overline{D}$ might never coincide with the runs where there are $k-1$ unique decisions in D. If we had a guarantee that all those executions, where the processes of $\overline{D}$ decide alone, can be extended in such a way that the processes of D reach $k-1$ unique decisions, however, we would be done. Indeed, the BRS-Theorem provides us with such a guarantee in a formal way by requiring that the runs where (dec-$\overline{D}$) holds are compatible for $\overline{D}$ with the runs where both (dec-D) and (dec-$\overline{D}$) hold.

It remains to investigate the formalization that consensus is impossible in $\overline{D}$. In order to do that, the theorem employs Definition 1. We could express that solving consensus using A is impossible on the subset $\overline{D}$ simply by stating that consensus is impossible in $\mathcal{M}' = \langle \overline{D} \rangle$, which is property (C) of the theorem. However, this is not sufficient because it only states that when considering $\overline{D}$ in isolation (without the processes $\Pi \setminus \overline{D}$ being present), consensus is impossible: In particular it does not exclude that the other processes might influence the processes in $\overline{D}$ via the system's synchrony conditions when running $\mathcal{A}$ on $\mathcal{M} = \langle \Pi \rangle$ in such a way that the processes in $\overline{D}$ are able to solve consensus.

The impossibility of consensus when running $\mathcal{A}_{|\overline{D}}$ on $\mathcal{M}'$ means that the set of all executions of $\mathcal{A}_{|\overline{D}}$ on $\mathcal{M}'$, denoted $\mathcal{M}'_{\mathcal{A}_{|\overline{D}}}$, contains at least one execution ρ' where some constraint of consensus is not met. If we could somehow guarantee that in the set of all executions of $\mathcal{A}$ on the larger system $\mathcal{M}$, denoted $\mathcal{M}_{\mathcal{A}}$, there is one execution ρ that is indistinguishable from ρ' to the processes of $\overline{D}$, we would be done: The processes that violate consensus in ρ' do so in ρ because they don't see a difference between ρ and ρ'. The theorem enforces this by property (D), which states that all executions of $\mathcal{M}'_{\mathcal{A}_{|D}}$ have one execution in $\mathcal{M}_{\mathcal{A}}$ that is indistinguishable for the processes in $\overline{D}$.

Additionally, property (D) reveals an interesting approach that addresses the problems that arise when dealing with system models $\langle \Pi \rangle$ and models of subsystems $\langle \overline{D} \rangle$ with $\overline{D} \subset \Pi$. When $\langle \overline{D} \rangle$ imposes some property on the executions of all algorithms (e.g. it makes consensus impossible), how can we transfer those executions to the larger system $\langle \Pi \rangle$? The issue at hand is that a consensus algorithm might include some state transitions that depend on the system

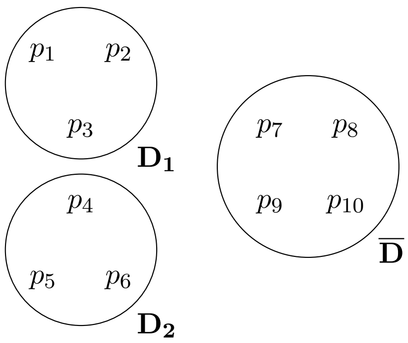


Figure 3.2: With up to $f = 7$ crash failures, all but the processes in D_1 or, alternatively, all but the processes in D_2 may crash. Another possibility is that all the processes in D_1 and D_2 plus one process in $\overline{D}$ may crash.

size; for an impossibility result, however, we want a predicate about any algorithm and therefore it would severely weaken the result when we state that the algorithm must not have any such dependencies. A solution for this somewhat intricate problem is revealed in the precise formulation of (D): We pick an arbitrary algorithm $\mathcal{A}$ for $\langle\Pi\rangle$ and construct $\mathcal{A}_{|\overline{D}}$ by restricting $\mathcal{A}$ to the processes of $\overline{D}$. By Definition 1, this implies that in $\mathcal{A}_{|\overline{D}}$ no messages are sent to any process outside of $\overline{D}$. In the next step, we run this restricted algorithm $\mathcal{A}_{|\overline{D}}$ on the smaller model $\langle\overline{D}\rangle$, which is possible because in $\mathcal{A}_{|\overline{D}}$ messages are only sent to processes in $\overline{D}$. The remaining step is to prove that all runs of $\mathcal{A}_{|\overline{D}}$ on $\langle\overline{D}\rangle$ have "matching" runs of $\mathcal{A}$ on $\langle\Pi\rangle$, which are indistinguishable for the processes of $\overline{D}$. This is expressed in a compact way by saying that the former are compatible with the latter, according to Definition 2. Once we are able to establish this compatibility, we indeed know that when the smaller model allows us to enforce properties on executions on any algorithm, including $\mathcal{A}_{|\overline{D}}$, then there exist executions of some algorithm on $\langle\Pi\rangle$, namely executions of $\mathcal{A}$ itself, where the processes of $\overline{D}$ exhibit the same behavior.

Apart from its significance for the BRS-Theorem, this is a very useful method when dealing with models of subsystems; we will make use of it again in Section 4.4.

3.3 Synchronous Processes with Asynchronous Messages

[8, Section 4] describes the application of the BRS-Theorem to a system with synchronous (and hence also asynchronous, c.f. Section 2.3) processes that communicate via asynchronous message passing. The number of allowed crash failures f is defined via the requirement

$$k \leq \frac{n-1}{n-f} \tag{3.1}$$

This failure assumption enables us to create a partitioning where one of the following failure patterns may occur in every run:

1. All processes of $\Pi \setminus D_j$ crash for some j, $1 \leq j < k$.

2. All processes of $\Pi \setminus \overline{D}$ crash and at least one process inside the partition $\overline{D}$ crashes.

A detailed proof of this statement can be found in [8, Theorem 2]; we illustrate it for the special case $n = 10$ and $k = 3$ in Figure 3.2. Note that for these values of n and k, the failure bound (3.1) implies we may have at least up to $f = 7$ crash failures.

By the asynchronous communication assumption, we may create runs where the messages between partitions are delayed arbitrarily long. Moreover, the processes of a partition D_j cannot wait for messages from any process $p \in \Pi \setminus D_j$, because p may have crashed according to the failure assumption. Thus, using unique proposal values x_i for each process p_i and delaying the messages accordingly leads to runs $\mathcal{R}_{(D,\overline{D})}$ that satisfy properties (A) and (B) of the theorem.

As the failure assumption allows at least one crash failure inside of $\overline{D}$ in every run, consensus is impossible in $\mathcal{M}' = \langle\overline{D}\rangle$ according to the FLP-result from [27]. Thereby, property (C) of the BRS-Theorem is also satisfied.

The remaining property (D) can be obtained by noting that the failure assumption supplies us with admissible runs of $\mathcal{M}_{\mathcal{A}}$ where all processes of $\Pi \setminus \overline{D}$ are initially dead. These runs are obviously indistinguishable for the processes of $\overline{D}$ from the runs $\mathcal{M}'_{A_{|\overline{D}}}$ where the processes in $\overline{D}$ don't receive any messages and any messages sent are only delivered to processes in $\overline{D}$.

We have hereby established conditions (A) - (D) of the BRS-Theorem. Note that we were able to derive the conditions merely by exploiting the asynchrony of the messages and the failure assumption (3.1), not depending on a specific algorithm in any way. We can therefore conclude from the application of the BRS-Theorem that solving k-set agreement in a system with synchronous processes, asynchronous messages and the failure bound (3.1) is impossible.

3.4 Wait-free k-Set Agreement with (Σ_k, Ω_k)

A more involved result, namely the impossibility of k-set agreement in a system with up to $(n-1)$ crash failures and failure detector (Σ_k, Ω_k), is obtained by another application of the BRS-Theorem in [8, Section 6].

The Failure Detector (Σ'_k, Ω'_k)

(Σ'_k, Ω'_k) is the *partition failure detector* from [8, Definition 6]. The history of (Σ'_k, Ω'_k) is called a *partitioning history* and because (Σ'_k, Ω'_k) is a combined failure detector, this history consists of a pair of values. While the value for Ω'_k is simply a legal history of Ω_k, the value Σ'_k is in essence a version of Σ_k that respects a partitioning: For every partition $D_i, 1 \leq i \leq k$, the history of Σ'_k at every correct process in D_i is a legal history of $\Sigma = \Sigma_1$ for the restricted model $\mathcal{M}_i = \langle D_i\rangle$. Additionally, once a process crashes, its history becomes the whole set Π rather than only the set of processes in its partition.

Comparing (Σ_k, Ω_k) and (Σ'_k, Ω'_k)

Recall that when we have an impossibility result for a failure detector $\mathcal{D}'$ and $\mathcal{D} \leq \mathcal{D}'$, then the impossibility also holds for $\mathcal{D}$. Since we want to show an impossibility for (Σ_k, Ω_k), it suffices to show it for (Σ'_k, Ω'_k) provided $(\Sigma_k, \Omega_k) \leq (\Sigma'_k, \Omega'_k)$ holds.

Showing $(\Sigma_k, \Omega_k) \leq (\Sigma'_k, \Omega'_k)$ can be reduced to showing that Σ'_k satisfies the properties of Σ_k because, by the definition of (Σ'_k, Ω'_k), we have $\Omega'_k = \Omega_k$.

Recall the definition of Σ_k from Section 2.5. Now, pick any $k + 1$ processes and examine their values for Σ'_k. As we have already observed, the liveness properties of Σ and Σ_k are identical, thus the liveness property of Σ_k is obviously satisfied by Σ'_k.

It remains to investigate the intersection property. Because we have only k partitions, any set of $k + 1$ processes includes at least two processes from the same partition. Their histories have, by the intersection property of Σ, at least one common element. Thus, we have that any set of $k + 1$ processes contains two processes with at least one common element in their values for Σ'_k, which matches precisely the definition of Σ_k.

Combining independent executions

Our goal is now to apply the BRS-Theorem in order to show that wait-free k-set agreement with (Σ'_k, Ω'_k) is impossible. In order to do so, however, we first require some results justifying the use of partitioning arguments in the presence of (Σ'_k, Ω'_k). As partitioning arguments rely on considering executions for isolated sets of processes, we must show that (Σ'_k, Ω'_k) does not provide some information about the situation outside a partition to nodes inside a partition in such a way that prohibits us from regarding them as isolated for our purposes. While the rigorous proofs for this fact can be found in [8, Lemma 6 and 7], we will provide only the basic idea here.

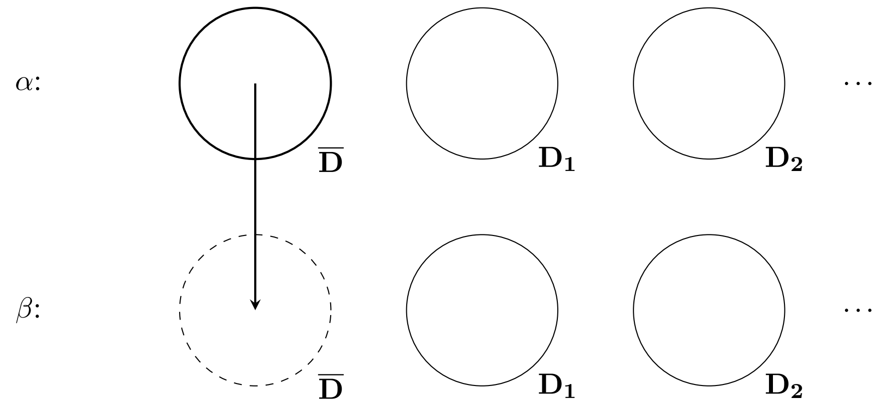


Figure 3.3: Given two executions α and β, where the nodes in $\overline{D}$ decide in isolation, can we replace the execution of $\overline{D}$ in β by $\overline{D}$ from α, resulting in some execution β', without the nodes in $\overline{D}$ noticing a difference?

We need two results: First, in a partitioned run β where the processes of a partition $\overline{D}$ decide isolated from the nodes of the other partitions, we want to be able to replace the execution of $\overline{D}$ in β with another execution of $\overline{D}$ in α, resulting in an execution β'. Moreover, the nodes of $\overline{D}$ should be unable to notice whether they are in α or in β' (Figure 3.3).
It turns out that such a replacement is indeed possible by letting $\overline{D}$ in β' receive the same messages, exhibit the same failure pattern, and see the same failure detector history as in α until

every correct process has decided in α and β. They key argument here is that we can do this because the history for $\overline{D}$ is not influenced by any process outside of $\overline{D}$ as, by definition of (Σ'_k, Ω'_k), Σ'_k respects only a quorum from the partition $\overline{D}$. Moreover, the history of Ω'_k at some process is independent of the history of Ω'_k at another process until t_{GST}, which we define as some time instant after every correct process has decided in α and β. After t_{GST}, we set the output of Ω'_k in β' to a set *LD* with *LD* $\cap\, (\Pi \setminus F_{\beta'}) \neq \emptyset$ in order to guarantee the (Eventual Leadership) property of Ω'_k in β'.

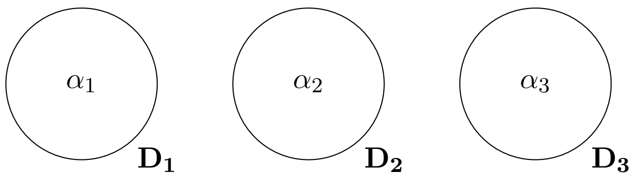


Figure 3.4: Given execution α_1 where all processes not in D_1 are dead initially (and analogously for α_2, α_3), can we combine them to an admissible execution $\alpha = \alpha_1 \circ \alpha_2 \circ \alpha_3$?

Second, we need to show that we can combine executions $\alpha_1, \alpha_2, \ldots$ of isolated partitions in a way that yields an admissible execution α for the entire system (Figure 3.4). With the reasoning from the previous point in mind, we can conclude that this is possible when we let $\alpha_1, \alpha_2, \ldots$ run simultaneously with the same failure patterns, message deliveries and failure detector histories for the respective partitions. Additionally, we delay the messages between the partitions until every correct process has decided. We choose as the history of Σ'_k in α the union of the histories Σ'_k in the isolated executions α_i. Finally, we define t_{GST} for Ω'_k in α as some time instant after all correct processes have decided and set the history of Ω'_k after t_{GST} in α to a set *LD* with *LD* $\cap\, (\Pi \setminus F) \neq \emptyset$.

Applying the BRS-Theorem

We now show that we can establish the four conditions of the BRS-theorem when solving k-set agreement in asynchronous systems with failure detector (Σ'_k, Ω'_k) for $2 \leq k \leq n-2$. We use the partitioning $D_i = \{p_i\}$ for $1 \leq i < k$ and $\overline{D} = \{p_k, \ldots, p_n\}$.

(A) $\mathcal{R}_{(\overline{D})}$ is nonempty:
Because we are allowed up to $n-1$ crash failures, the execution where all but the processes in $\overline{D}$ are initially dead, is admissible.

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$:
As indicated by the explanations throughout this section, we can combine executions of single partitions, yielding a run $\beta \in \mathcal{R}_{(D,\overline{D})}$. In such a run β, we may replace the execution of the processes in $\overline{D}$ by the execution of the processes of $\overline{D}$ in α, yielding some run $\beta' \in \mathcal{R}_{(D,\overline{D})}$ with $\alpha \overset{\overline{D}}{\sim} \beta'$.

(C) Consensus is impossible in $\mathcal{M}' = \langle \overline{D} \rangle$:
Note that since we are only interested in $2 \leq k \leq n-2$, we have, because $\overline{D} =$

$\{p_k, \ldots, p_n\}$, $|\overline{D}| \geq 3$. We are interested in the restricted model $\mathcal{M}' = \langle \overline{D} \rangle$ and need to investigate what failure detector is available to the processes in $\mathcal{M}'$. By $|\overline{D}| \geq 3$ and the definition of (Σ'_k, Ω'_k), the strongest failure detector that is possibly accessible in $\mathcal{M}'$ is (Σ, Γ), where Γ outputs arbitrary processes during its anarchy period until it stabilizes. After stabilization, Γ outputs the same constant set *LD*, containing at least one correct process, with $|LD \cap \overline{D}| = 2$.

We observe that Γ is equivalent to Ω_2: We remove the processes not in $\overline{D}$ from the output of Ω'_k at any process in $\overline{D}$, and add processes arbitrarily from that D in order to arrive at Γ. Note that Γ eventually contains at least one correct process $\in \overline{D}$ since Ω'_k does so. Because (Σ, Ω_2) is strictly weaker than (Σ, Ω) (cf. [28, 37]) and (Σ, Ω) is the weakest failure detector for solving wait-free consensus [29], consensus is impossible in $\mathcal{M}'$.

(D) $\mathcal{M}'_{A_{|\overline{D}|}} \preccurlyeq_{\overline{D}} \mathcal{M}_A$:
Take any run $\rho \in \mathcal{M}'_{A_{|\overline{D}|}}$. As we have seen from the arguments made throughout this section — in particular, regarding the partitioning property of the history of (Σ'_k, Ω'_k) — it is possible to find a run $\rho' \in \mathcal{M}_A$ where the processes in D are initially dead and the processes in ρ' make the same state transitions as in ρ.

This successful application of the BRS-theorem, in conjunction with the relation $(\Sigma_k, \Omega_k) \leq (\Sigma'_k, \Omega'_k)$, yields the impossibility of wait-free k-set agreement in asynchronous message passing systems with failure detector (Σ_k, Ω_k). Together with the possibility of 1-set agreement (consensus) with (Σ_1, Ω_1) [28] and $(n-1)$-set agreement with $(\Sigma_{n-1}, \Omega_{n-1})$ [13] we have the following corollary:

Corollary 1 (From [8, Corollary 2]). *There is an $(n-1)$-resilient algorithm that solves k-set agreement with failure detector class $(\Sigma_k, \Omega_k)_{1 \leq k \leq n-1}$ in an asynchronous system, if and only if $k = 1$ or $k = n-1$.*

3.5 The Search for a Weakest Failure Detector for Message Passing k-Set Agreement

In Corollary 1, we have seen that the failure detector $(\Sigma_k, \Omega_k)$is too weak to solve general k-set agreement in message passing systems for $1 < k < n-1$. In an attempt to find a failure detector that enables solving k-set agreement in message passing system, the $(n-k)$*-loneliness detector* $\mathcal{L}_k$ was introduced in [11]. In contrast to the previously mentioned failure detectors, $\mathcal{L}_k$ supplies a boolean variable. This variable is FALSE at $n-k$ processes at all times, while it becomes TRUE at a correct process in the event that k or more processes are faulty. Thereby, it detects the case where $n-k$ or less processes are "lonely", allowing an algorithm to potentially circumvent the impossibility for this scenario. $\mathcal{L}_k$ is formally defined as follows:

Definition 3 ([11, Definition 1]). *The $(n-k)$-loneliness detector $\mathcal{L}_k$ outputs* TRUE *or* FALSE, *such that for all environments $\mathcal{E}$ and $\forall F \in \mathcal{E}$ it holds that there is a set of processes $\Pi_0 \subseteq$*

$\Pi, |\Pi_0| = n - k$ *and a correct process q such that:*

$$\forall p \in \Pi_0 \forall t \colon H(p,t) = \text{FALSE} \tag{3.2}$$

$$|F| \geq k \Rightarrow \exists t \forall t' \geq t \colon H(q,t') = \text{TRUE} \tag{3.3}$$

Although an algorithm exists that uses $\mathcal{L}_k$ in order to solve k-set agreement in message passing systems [11], it has been shown that $\mathcal{L}_k$ is not the weakest failure detector for this problem. But how does $\mathcal{L}_k$ compare directly to (Σ_k, Ω_k)? Recently, this question has been answered, as failure detector X_k that is stronger than Ω_k was presented in [36]. It has the following interesting property: (Σ_k, X_k) is equivalent to $\mathcal{L}_k$.

While Σ_k allows the system to partition it at most k parts, it is the conjunction of Σ_k and X_k that provides a *perpetual* property that allows us to solve consensus in these partitions. In contrast, we have seen in the previous section that Ω_k is not strong enough to solve consensus in every partition.

Obviously, another failure detector that is (slightly) stronger than Ω_k but weaker than X_k would be a promising candidate for the still unknown weakest failure detector for k-set agreement in message passing systems for $1 < k < n-1$. The BRS-Theorem seems to be a promising tool for supporting this endeavor, as it is tailored to exactly these kinds of models and abstracts away from the complexities involved when proving impossibilities in such settings. Following this road will be an important direction for further research.

CHAPTER 4

Directed Dynamic Networks

In this chapter, we investigate k-set agreement in systems with highly dynamic communication. Such scenarios occur e.g. in wireless networks, where a node p may be reachable from a node q only via intermediate nodes for a certain period, or even worse be disconnected entirely for some time. We first present a model for such networks where communication is according to a sequence of round-by round graphs and discuss some known results regarding the solvability of consensus in this model. Subsequently, we show fundamental limitations to which nodes in these dynamic systems are subject to and present some new results regarding the impossibility of k-set agreement in those systems.

4.1 Model Overview

In the directed dynamic network (DDN) model from [10], processes are synchronous and operate in lock-step rounds, while communication is modeled via a sequence of round-by-round communication graphs $\mathcal{G}^r$, $r \geq 1$. The round r communication graph $\mathcal{G}^r = \langle V^r, E^r \rangle$ is a directed graph, consisting of a set of nodes V^r and a set of edges E^r. For simplicity we write $p \in \mathcal{G}^r$ instead of $p \in V^r$ and $(p \to q) \in \mathcal{G}^r$ instead of $(p \to q) \in E^r$. Each process $p \in \Pi$ corresponds to a node of the same name $p \in \mathcal{G}^r$. An edge $(p \to q)$ is in $\mathcal{G}^r$ if and only if p can send a message to q in round r. Messages in this model are confined to communication closed rounds, i.e. a message sent in round r may only be received in round r.

Since our algorithms are synchronous and deterministic, this implies that, for a certain algorithm and initial configuration, each execution is uniquely determined by the sequence of communication graphs for each round.

A central aspect of the DDN model are root components. A round r root component is a strongly connected component $\mathcal{R}^r$, where no node has an incoming edge from a node that is not in $\mathcal{R}^r$.

Definition 4 (From [10, Section 4]). *We call a strongly connected component $\mathcal{R}^r$ a* ***round r root component*** *if*
$\forall p \in \mathcal{R}^r \; \forall q \in \mathcal{G}^r \colon (q \to p) \in \mathcal{G}^r \Rightarrow q \in \mathcal{R}^r$

A property of the communication graph especially important for k-set agreement is the maximum number of root components $\mathcal{G}^r$ contains. Consider for example the case, where each $\mathcal{G}^r$ may contain more than k root components. In this instance we may construct an execution where each round-by round communication graph contains the same $k+1$ nodes as single-node root components. When starting from an initial configuration where each process holds a unique input value, the $k+1$ processes that constitute the $k+1$ root components must, by the termination condition, decide eventually. Because as root components they never received any messages, they must decide on their own input value. This leaves us with $k+1$ distinct decision values which violates the safety property *k-agreement*. Thus we have found an execution where every k-set agreement algorithm fails to solve the problem and thereby shown that k-set agreement is impossible in the DDN model if the communication graph is allowed to contain more than k root components. Note that this implies the impossibility of consensus with more than a single root component.

We will now introduce two concepts that enable a precise reasoning about DDNs, followed by a theorem which establishes an impossibility of consensus in DDNs on which we will rely subsequently.

First, we consider root components that last for multiple rounds (c.f. [10, Section 4]). Let $I = [r_i, r_{i+l}]$ be the interval of rounds of length $l+1 = |I|$, which spans from the beginning of round r_i to the end of round r_{i+l}. We call a root component $\mathcal{R}^I$ *I-vertex stable* if the processes in $\mathcal{R}^I$ form a root component in all rounds of I. Note that this includes the possibility that the topology of $\mathcal{R}^{r_i}$ changes during I, as long as the nodes in $\mathcal{R}^{r_i}$ remain a root component.

Second, we introduce influence chains (c.f. [10, Section 3.3]). A node p is said to influence a node q in round r, in symbols $p \overset{r}{\rightsquigarrow} q$, if q received a message from p in round r, i.e. if $(p \to q) \in \mathcal{G}^r$ or if $p = q$ (we assume that nodes receive their own messages).
We say that there is a causal influence chain of length l from p to q which starts in round r, in symbols $(p \overset{r[l]}{\rightsquigarrow} q)$, if there exists a sequence of $l+1$ (not necessarily distinct) processes $p = p^0, \ldots, p^l = q$ such that $(p^i \overset{r+i}{\rightsquigarrow} p^{i+1})$, $0 \leq i < l$. Intuitively, $(p \overset{r[l]}{\rightsquigarrow} q)$ corresponds to a path through $\mathcal{G}^{[r,r+l]}$ from p to q, where in each round we stay at the same node or move along one outgoing edge.
We can now define the *causal distance* $\texttt{cd}^r(p,q)$, which is the length of the shortest causal influence chain from p to q that starts in round r: $\texttt{cd}^r(p,q) := \min\{l \mid (p \overset{r[l]}{\rightsquigarrow} q)\}$.

The following impossibility result states that consensus is impossible in a system with a single root even if it is vertex-stable for an interval of length D and there is just one process which is so far away from the root that it can receive not a message from the root within D rounds.

Assumption 1 (from [10, Assumption 4]). *For any round r, there is exactly one root component $\mathcal{R}^r$ in $\mathcal{G}^r$. Moreover, there exists a D and an interval of rounds $I = [r_{ST}, r_{ST} + D - 1]$, such* *that there is an I-vertex stable root component $\mathcal{R}^I$, and there exists a unique $q \in \Pi$ such that $\forall p \in \mathcal{R}^I, \forall r \in I\colon \mathtt{cd}^r(p,q) \leq D+1$, while for all $q' \in \Pi \setminus \{q\}$ we have $\forall p \in \mathcal{R}^I, \forall r \in I\colon \mathtt{cd}^r(p,q') \leq D$.*

Theorem 2 (from [10, Theorem 5])**.** *Assume that Assumption 1 is the only requirement for the graph topologies. Then consensus is impossible.*

Theorem 2 gives a lower bound on the duration a root component must remain stable in order for consensus to (potentially) be solvable. It states that if it takes D rounds for a message from any process of the root component to reach some unique process q, then consensus cannot be solved if the root component is stable for merely $D-1$ rounds. Note that the proof of Theorem 2 actually implies that even if there are multiple distinct roots, each stable for less than D rounds in a single run, consensus cannot be solved.

Informally, this impossibility of binary consensus (and hence consensus) under Assumption 1 results from the following fact: In a system adhering to the DDN model that satisfies only Assumption 1, two univalent round r configurations C' and C'' that differ only in the state of a single process p cannot have a different valence (cf. [10, Lemma 16]). This is the case because, under Assumption 1, there are configurations reachable from both C' and C'', by a distinct sequence of round graphs, where some process q decides without knowledge of the state of p – and thus on the same value. The following cases illustrate this:

1. If p is part of the root component $\mathcal{R}^I$, Assumption 1 allows us to find some q which is not influenced by p (particularly if $r \in I$). When the sequence of round graphs is such that q becomes a single node root component in all rounds after I ends, q will eventually decide independent of p's state.

2. If p is not part of the root component $\mathcal{R}^I$ and some q that is part of the root component in round r becomes the single node root component in all subsequent round graphs, q will decide eventually, independent of p's state.

The fact that the state of a single process cannot determine whether a configuration is 0-valent or 1-valent implies that when two configurations differ only in the state of a single process, either both of them have the same valence or one of them is bivalent. This in turn allows us to show the existence of an execution that never terminates, by staying bivalent forever, as described subsequently. First, it is shown that a bivalent initial configuration exists, followed by a justification why every bivalent configuration has a bivalent successor configuration.

A bivalent initial configuration exists, because from the (by validity) 0-valent initial configuration where all proposal values $x_i = 0$, by subsequently setting one proposal value to 1, we will eventually reach the (by validity) 1-valent initial configuration where all proposal values are 1. Therefore, at some point, the valence of the configuration changes when setting a proposal value of a single process. As the change of a single proposal value in an initial configuration is equivalent to the change of the state of a single process in this configuration, by the previous argumentation, it is impossible that the change occurs from 0-valent to 1-valent. Hence it must occur from 0-valent to bivalent (and some point from bivalent to 1-valent), which confirms the existence of a bivalent initial configuration.

Every bivalent round r configuration C has a bivalent successor configuration for a similar reason: As we have already mentioned, in the DDN model, since computations are deterministic, given some configuration C, we may uniquely determine a successor configuration by its corresponding round graph. We proceed by assuming, for the purpose of deriving a contradiction, that all successor configurations of C are univalent. By the bivalence of C, there must be a successor configuration of C that resulted by applying some communication graph $\mathcal{G}'$, denoted $C(\mathcal{G}')$, that is 0-valent and some successor configuration $C(\mathcal{G}'')$ that is 1-valent. The key argument is that one can construct a sequence of graphs $\mathcal{G}' = \mathcal{G}_0, \mathcal{G}_1, \ldots, \mathcal{G}_k = \mathcal{G}''$ when $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ for $0 \leq i < k$ differ in a single edge only, which is known as *connectedness of the successor graphs*: We can incrementally add an edge to $\mathcal{G}'$, first by merging the root components of $\mathcal{G}'$ and $\mathcal{G}''$ and subsequently adding the remaining edges, to arrive at $\mathcal{G}' \cup \mathcal{G}''$. Then we may incrementally remove edges from $\mathcal{G}' \cup \mathcal{G}''$, first by shrinking the root of $\mathcal{G}' \cup \mathcal{G}''$ and subsequently removing the remaining edges, to arrive at $\mathcal{G}''$.

Since we assumed that $C(\mathcal{G}')$ is 0-valent and $C(\mathcal{G}'')$ is 1-valent, at some point in this sequence, the round graphs $\mathcal{G}_i$ and $\mathcal{G}_{i+1}$ must have caused a valency switch in the corresponding configurations $C(\mathcal{G}_i)$ and $C(G_{i+1})$. Since we know that this switch could not have been from 0-valent to 1-valent, because the two respective configurations differ only in the state of a single process (which is equivalent to a single edge in the communication graph), it must have been from 0-valent to bivalent.

4.2 Impossibility of Consensus with Lossy Links

The DDN model allows us to express link failures in a precise manner: From the point of view of a process p we have that p may call the message sending function in order to send a message to another process q in round r but if there is no edge $(p \to q)$ in $\mathcal{G}^r$, the message simply won't be delivered. This behavior can be characterized as a lossy link and for this reason we will repeat a theorem about the impossibility of consensus in two-process systems with lossy links, which will prove very useful in the subsequent section.

The well-known Gray's Theorem [30] states that it is impossible to solve the coordinated attack problem, which is very similar to consensus, in a two-process system with arbitrarily lossy links. A generalization of Gray's theorem has been introduced in [44]. We show that this result is also valid in the DDN model:

Theorem 3. *There is no deterministic algorithm that solves consensus in the DDN model with two processes connected by a lossy link, even if communication is reliable in one direction in every round.*

Proof. Similar to the strategy used in [44] to show the undecidability with lossy links, our proof proceeds by induction.

For the base case consider the initial configuration $C^1(x_1, x_2)$, where p_1 starts with initial value x_1 and p_2 starts with x_2 and $x_1 \neq x_2$. Assume that $C^1(x_1, x_2)$ is univalent. By Validity the only possible decision values in runs starting from this initial configuration are x_1 and x_2. In order to see that $C^1(x_1, x_2)$ is neither x_1-valent nor x_2-valent (and hence bivalent) we show that $C^1(x_1, x_2)$ is not x_1-valent (the case of x_2-valency follows from the symmetric argument).

Consider a run starting from $C^1(x_1, x_2)$ where $\forall r > 0\colon (p_1 \to p_2) \notin \mathcal{G}^r$. This run is indistinguishable to p_2 from a run with the same communication graphs but starting from $C^1(x', x_2)$, with $x' \neq x_1$. By Validity p_2 cannot decide x_1 in the latter run, showing that $C^1(x_1, x_2)$ cannot be x_1-valent.

For the inductive step, we show that if C^{r-1} is bivalent then there is a bivalent successor configuration C^r of C^{r-1} that is bivalent.

Assume that all C^r are univalent. As the successor configurations of C^{r-1} are uniquely determined by the round graph $\mathcal{G}^{r-1}$ and because of the assumption that there is a single root component, we need to consider only three successor configurations. Let C^r_{01} be the successor configuration of C^{r-1} that is reached by the $\mathcal{G}^{r-1}$ with $E^{r-1} = \{(p_1 \to p_2)\}$, let C^r_{10} be the successor configuration of C^{r-1} that is reached by the $\mathcal{G}^{r-1}$ with $E^{r-1} = \{(p_1 \leftarrow p_2)\}$, and let C^r_{11} be the successor configuration of C^{r-1} that is reached by the $\mathcal{G}^{r-1}$ with $E^{r-1} = \{(p_1 \to p_2), (p_1 \leftarrow p_2)\}$. As all C^r are univalent, w.l.o.g. assume that C^r_{11} is x_1-valent. Because C^{r-1} is bivalent, at least one of C^r_{10}, C^r_{01} must be x_2-valent; w.l.o.g. assume that C^r_{10} is x_2-valent. Note that the only difference between C^r_{11} and C^r_{10} is that p_2 received p_1's message in the former but not in the latter. Consider now the executions starting from C^r_{10}, respectively C^r_{11}, where it holds that $\forall r' > r\colon (p_1 \leftarrow p_2) \notin \mathcal{G}^{r'}$. Both executions are indistinguishable for p_1 because p_2 can never tell p_1 whether p_2 received p_1's round r' message. Since p_1 must eventually decide the same in both executions, they cannot have different valences.

□

It is important to note that if the pattern of link failures is known in advance to the algorithm, then it is possible to solve consensus when communication is reliable in one direction every round. Moreover, for Theorem 3 to hold, it is required that the successor graphs are connected, i.e., that the possible communication patterns for each round are from a set of directed graphs where for each graph H in this set there is another graph J in this set such that H and J differ only in one edge. A rigorous notation of "withholding information", which captures the required properties can be found in [44]. The key issue to note here is that the uncertainty of the link failure pattern makes it possible for the adversary to force an execution that does not terminate, for any algorithm.

Using Theorem 3 as well as the notion of restrictions of algorithms and compatibility of runs from Definitions 1 and 2, we are now able to prove the impossibility of consensus in the DDN-model with moving root components.

Theorem 4. *Consensus is impossible in the DDN model for $n \geq 2$ if the communication graph contains a single root component that may move arbitrarily in every round.*

Proof. For the purpose of deriving a contradiction, assume that an algorithm $\mathcal{A}$ exists that solves consensus under these assumptions. Let $D = \{p, q\} \subseteq \Pi$, let $\mathcal{M}' = \langle D \rangle$ and let $\mathcal{M}'_{\mathcal{A}_{|D}}$ be the set of runs of $\mathcal{A}_{|D}$ on $\mathcal{M}'$. Consider the runs $\mathcal{H}$ of $\mathcal{A}$ on $\langle \Pi \rangle$, where for the root component $\mathcal{R}^r$ in every round $r > 0$, it holds that $\mathcal{R}^r \subseteq \{p, q\}$. By the assumptions of the theorem, $\mathcal{H}$ is nonempty. Observe that $\mathcal{M}'_{\mathcal{A}_{|D}} \preccurlyeq_D \mathcal{H}$ holds: Since, in $\mathcal{H}$, neither p nor q ever receive a message from a process of $\Pi \setminus D$, it is easy to find for any $\rho \in \mathcal{M}'_{\mathcal{A}_{|D}}$ a matching run $\rho' \in \mathcal{H}$ s.t. $\rho \overset{D}{\sim} \rho'$ is established. By Theorem 3, $\mathcal{M}'_{\mathcal{A}_{|D}}$ contains at least one run σ such that consensus is not solved in σ. Thus, by the compatibility of $\mathcal{M}'_{\mathcal{A}_{|D}} \preccurlyeq_D \mathcal{H}$, the set of runs $\mathcal{H}$, and thereby the set of runs of $\mathcal{A}$ on $\langle \Pi \rangle$, contains a run where consensus is not solved in D. Since $\mathcal{A}$ solves consensus system-wide, it also solves consensus among the processes of D – a contradiction. □

Note that Theorem 4 could also have been immediately derived from Theorem 2: After all, not having to have a root component that is vertex-stable for more than a single round at all is weaker than Assumption 1. Therefore, Theorem 2 is also applicable in this case. This is not surprising: Theorem 2 internally relies on an equivalent argument as Theorem 3, on which in turn Theorem 4 is based. Nevertheless, the proof of Theorem 4 shows how we can directly reduce the impossibility of consensus in a larger system in the DDN setting to the impossibility of consensus in a small system.

We have now provided all the impossibilities for consensus on which we rely on in the subsequent sections, where we provide some new results about the impossibility of k-set agreement in directed dynamic networks.

4.3 Necessity of Simultaneously Static Root Components for k-Set Agreement

The results from the next two sections have been devised in collaboration with Manfred Schwarz, hence can be found in a somewhat different form also in his Master's thesis [45] (and in a joint publication [46] that is currently under review). The latter also presents assumptions that suffice to solve k-set agreement in the DDN model as well as an according algorithm.

The first assumption we introduce in our model to exclude trivial impossibilities, as discussed in Section 4.1, is a restriction on the total number of root components per round:

Assumption 2. *$\forall r > 0$, $\mathcal{G}^r$ contains at most k root components.*

We now use the BRS-Theorem to show that, in order to solve k-set agreement, it does not suffice to have k root components simultaneously for only one round. The following theorem shows that k root components need to be static for some time, in order to be able to solve k-set agreement.

Theorem 5. *There exists no algorithm that solves k-set agreement in the directed dynamic network model with $n > k$ nodes and k root components, if one of the k root components may change arbitrarily every round.*

Proof. Suppose that there is a k-set algorithm $\mathcal{A}$ that works correctly under the assumptions of our theorem. We will prove that the conditions of the BRS-Theorem (Theorem 1) are satisfied, thereby providing a contradiction to the assumption that $\mathcal{A}$ exists. Let $D_i = p_i$ for $0 < i \leq k-1$ and let $D = \bigcup_{i=1}^{k-1} D_i$. Consequently, $\overline{D} = \{p_k, p_{k+1}, \dots, p_n\}$ and $|\overline{D}| \geq 2$.

(A) $\mathcal{R}_{(\overline{\mathbf{D}})}$ is nonempty:

In order to create an execution where the processes in $\overline{D}$ do not receive any messages from the processes of D, we let the communication graph in every round not have any incoming links to $\overline{D}$ from D until every process in $\overline{D}$ has decided. Clearly such a sequence of communication graphs constitutes an admissible execution, as it satisfies the assumptions of the theorem. This establishes $\mathcal{R}_{(\overline{\mathbf{D}})} \neq \emptyset$.

(B) $\mathcal{R}_{(\overline{\mathbf{D}})} \preccurlyeq_{\overline{\mathbf{D}}} \mathcal{R}_{(\mathbf{D},\overline{\mathbf{D}})}$:
Let $\mathcal{H}$ be the set of runs where processes p_i have unique input values $x_i = i$, $0 < i < k$, the communication graph in every round is such that $p_1, \dots, p_{k-1}$ are disconnected, and $p_k, \dots, p_n$ are weakly connected until every process has decided. By the assumptions of our theorem, $\mathcal{H}$ is non-empty.
Since the processes of $\overline{D}$ never receive a message from a process of D in both $\mathcal{R}_{(\overline{D})}$ and $\mathcal{H}$ and, moreover, the initial values of $\overline{D}$ are not restricted in $\mathcal{H}$ in any way, it is easy to find for any run $\rho \in \mathcal{R}_{(\overline{D})}$ a run $\rho' \in \mathcal{H}$, s.t. $\rho \overset{\overline{D}}{\sim} \rho'$.
Because obviously $\mathcal{H} \subseteq \mathcal{R}_{(D,\overline{D})}$, we have established $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$.

(C) Consensus is impossible in $\mathcal{M}' = \langle \overline{\mathbf{D}} \rangle$:
We let the processes in $\overline{D}$ be weakly connected and have one root component in $\overline{D}$ in each round. Because $|\overline{D}| \geq 2$, we may have a moving root component every round, so the processes of $\overline{D}$ solving consensus would be a contradiction to Theorem 4.

(D) $\mathcal{M}'_{\mathcal{A}_{|\overline{\mathbf{D}}|}} \preccurlyeq_{\overline{\mathbf{D}}} \mathcal{M}_{\mathcal{A}}$:
Fix any run $\rho' \in \mathcal{M}'_{\mathcal{A}_{|\overline{D}|}}$ and consider the run $\rho \in \mathcal{M}_{\mathcal{A}}$ where every process in $\overline{D}$ has the same sequence of states in ρ as in ρ'. By the properties of $\mathcal{M}$, the processes in D can be disconnected in every round of ρ, yielding $\rho \overset{\overline{D}}{\sim} \rho'$.

□

When examining what we did in Theorem 5 more closely, we find that the communication graph always allows us to make the partitioning, as required by the BRS-Theorem, under the assumption that we may have up to k root components. This is interesting, because it means that applying the BRS-Theorem in this setting can be reduced to finding the impossibility for consensus in $\overline{D}$. In other words, when we have an impossibility for consensus for $n - k + 1$ processes and Assumption 2 is the only assumption that holds in our system, we may immediately infer an impossibility for k-set agreement by referencing the BRS-Theorem. For instance, we may use Theorem 2 to further strengthen Theorem 5 for $n > k + 1$:

Theorem 6. *There exists no algorithm that solves k-set agreement with $n > k + 1$ processes under Assumption 2, for any $1 \leq k < n$, even if there are $k - 1$ root components R_1 to R_{k-1} that are vertex-stable forever and one root component R_k is vertex-stable for at most $n - k - 1$ rounds.*

Proof. For $k = 1$, Theorem 6 is equivalent to Theorem 2 for $D = n-k-1 = n-2$. To prove the theorem for $k > 1$, we show again that the conditions of the BRS-Theorem (Theorem 1) apply. As the system model is the same as in the proof of Theorem 5, we use the same partitioning once again. Thus, the proof of conditions (A), (B) and (D) remains unchanged and it suffices to show that condition (C) is satisfied:

(C) Consensus is impossible in $\mathcal{M}' = \langle \overline{\mathbf{D}} \rangle$: Recall that $\overline{D}$ is the partition of the k^{th} root component $\mathcal{R}_k$, which is perpetually changing every round, except for some interval of rounds

$I = [r_{ST}, r_{ST} + \ell - 1]$, where $\ell = n - k - 1$, for some fixed r_{ST}. During this interval, let the topology of $\overline{D}$ be such that there exists some $p \in \mathcal{R}_k$ and some $q \in \overline{D}$ with $\mathtt{cd}^{r_{ST}}(p, q) = n - k$. Since $|\overline{D}| = n - k + 1$, such a topology (e.g. a chain with head p and tail q) adhering to the conditions of Theorem 2 for $D = n - k - 1$ exists. Hence, consensus is impossible in $\overline{D}$.

$\square$

4.4 Impossibility of k-Set Agreement with Decision Hiding

By applying the BRS-Theorem to the DDN model, Theorem 5 and Theorem 6 established that we need k root components that hold simultaneously for a certain interval in order to solve k-set agreement. It is not trivial to find a tight lower bound for how long these root components must hold for k-set agreement to become solvable. Despite this, an algorithm that solves k-set agreement should certainly exist when the root components are guaranteed to be stable for a very long period of time. Does this mean that if, after some global stabilization round r_{GST}, there exist k static simultaneous root components forever, k-set agreement is always solvable? Maybe surprisingly, without additional restrictions, this is not generally the case. The reasons for this will be examined in the following Theorem 7, which shows that it is possible to construct an execution where the safety property *k-agreement* is violated.

Theorem 7. *Suppose that in every run there is a* stabilization round r_{GST} *such that, for all $r \geq r_{\text{GST}}$, it holds that $\mathcal{G}^r = \mathcal{G}^{r+1}$ and there are no other restrictions on the communication graphs apart from Assumption 2. Then, there is no algorithm that solves k-set agreement for $1 < k < n$.*

Proof. We start our proof with some notation and technical lemmas. For some model $\mathcal{M}$ and some algorithm $\mathcal{A}$, we denote by $\mathcal{M}_{\mathcal{A}}$ the set of runs of algorithm $\mathcal{A}$ on $\mathcal{M}$. Let $\mathcal{M} = \langle \Pi \rangle$ be our system with $|\Pi| > 2$ that is restricted by the assumptions of the theorem, and let $D = \{p_1, p_2\} \subseteq \Pi$. We consider the restricted model $\mathcal{M}' = \langle D \rangle$ and the restricted algorithm $\mathcal{A}_{|D}$ of algorithm $\mathcal{A}$ on D. Except for the number of processes, $\mathcal{M}'$ has the same properties as $\mathcal{M}$, except that $\mathcal{M}'$ guarantees a single root component in every round.

Following the general idea of the BRS-Theorem (Theorem 1), we will argue that if there was a correct k-set agreement algorithm $\mathcal{A}$ for $\mathcal{M}$, then the restriction $\mathcal{A}_{|D}$ would solve consensus when being run on $\mathcal{M}'$: Since the assumption of k root components per round allows $\mathcal{G}_r$ to partition into k isolated partitions with a single root each (subsequently referred to as root partitions), there are executions in $\mathcal{M}_{\mathcal{A}}$ where the processes in D receive no messages from any process outside of D, and decide on a unique value value in every root partition. On the other hand, when executing $\mathcal{A}_{|D}$ on $\mathcal{M}'$, the processes of D clearly also receive no messages from any process of $\Pi \setminus D$. Thus, the processes of D cannot distinguish whether they execute $\mathcal{A}_{|D}$ on $\mathcal{M}'$ or $\mathcal{A}$ on $\mathcal{M}$, and must hence also agree on a single value. Note that, because $n = |D| = 2$ in $\mathcal{M}'$, we can re-use classic bivalency arguments since there are at most two initial values (although we assume $|V| > k$ for k-set agreement, as usual).

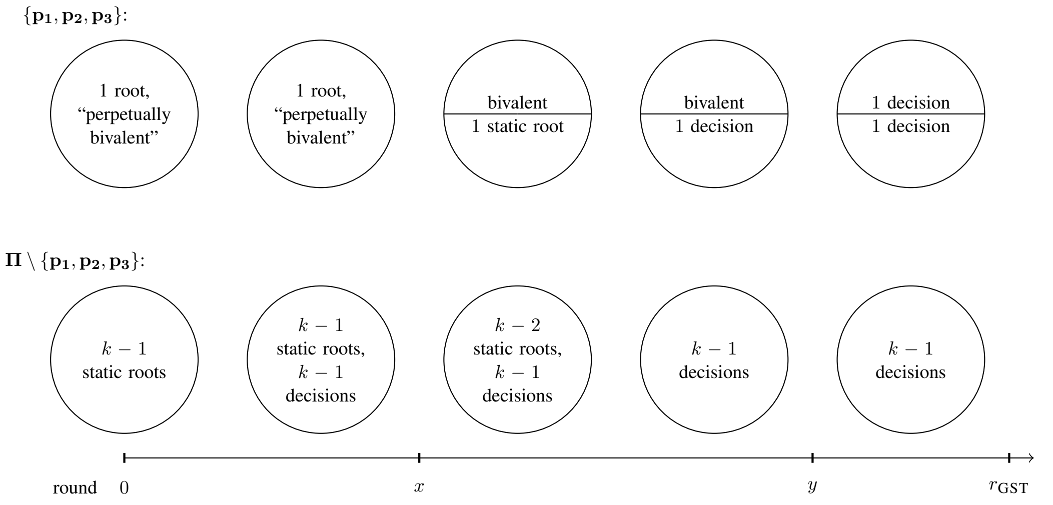
We commence the proof by establishing the following technical lemmas.

Lemma 1. *For every k-set agreement algorithm $\mathcal{A}$ for $\mathcal{M}$, there exists a run $\rho' \in \mathcal{M}'_{\mathcal{A}|_D}$ that is bivalent for all rounds up to and including r_{GST}.*

Proof. Since $|\mathcal{M}'| = 2$, solving non-trivial k-set agreement in $\mathcal{M}'$ corresponds to solving consensus in $\mathcal{M}'$. The existence of a bivalent run follows immediately from the proof of Theorem 3. □

Lemma 2. *For every k-set agreement algorithm $\mathcal{A}$ for $\mathcal{M}$, the set of runs $\mathcal{R} \subseteq \mathcal{M}_A$ where $\mathcal{G}^r$ contains arbitrary outgoing edges from D but no incoming edges to D satisfies $\mathcal{M}'_{\mathcal{A}|_D} \preccurlyeq_D \mathcal{R}$.*

Proof. By the assumptions of $\mathcal{R}$, the processes in D never receive messages from any process of $\Pi \setminus D$. Therefore, in any run of $\mathcal{R}$, the state transitions of the processes in D cannot depend on the state of any process of $\Pi \setminus D$. This establishes $\mathcal{M}'_{\mathcal{A}|_D} \preccurlyeq_D \mathcal{R}$. □

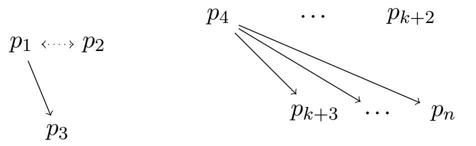


$\{p_1, p_2, p_3\}$:
1 root, "perpetually bivalent"
1 root, "perpetually bivalent"
bivalent
1 static root
bivalent
1 decision
1 decision
1 decision
$\Pi \setminus \{p_1, p_2, p_3\}$:
$k-1$ static roots
$k-1$ static roots, $k-1$ decisions
$k-2$ static roots, $k-1$ decisions
$k-1$ decisions
$k-1$ decisions
round
0
x
y
r_{GST}

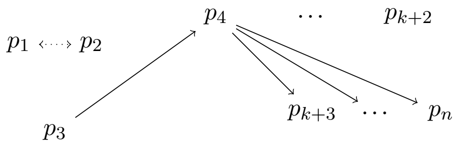
Figure 4.1: Overview of the proof of Theorem 7.

We are now ready to prove Theorem 7. Our proof relies on an execution, where every k-set agreement algorithm with $n > 2$ and $1 < k < n - 1$ produces $k + 1$ decisions. The run is constructed as follows (cf. Fig. 4.1 for an overview): For each p_i, we choose a unique proposal value x_i such that x_1 and x_2 are in accordance with Lemma 1. For the rounds $1 \leq r \leq x$, where x is chosen as described below, we use a graph $\mathcal{G}^r$ constructed as follows (cf. Fig. 4.2a):

- p_1, p_2 are connected to each other as in the bivalent run ρ' provided by Lemma 1, and have no incoming edges from any $\Pi \setminus D$.
- p_3 has an incoming edge only from p_1 and no outgoing edges.
- $p_4, \ldots, p_{k+2}$ form single-node root components.



(a) One (changing) root component among p_1 and p_2, single root p_4, $k-2$ remaining single-node root components $p_5, \ldots, p_{k+1}$ among $p_5, \ldots, p_n$.



(b) One root component among p_1 and p_2, single root p_3, $k-2$ remaining single-node root components $p_5, \ldots, p_{k+1}$ among $p_5, \ldots, p_n$.

Figure 4.2: Communication graphs used in the proof of Theorem 7: (a) depicts $\mathcal{G}^r$ for $0 < r \leq x$, while (b) depicts $\mathcal{G}^r$ for $x < r \leq y$. The dotted edge indicates an unstable ("moving") link between p_1 and p_2.

- The remaining processes (if any) have an incoming edge from p_4 but no outgoing edges.

Note that this graph contains one root partition $\{p_1, p_2, p_3\}$ and $k-1$ other root partitions with singleton root components $p_4, \ldots, p_n$, thereby satisfying Assumption 2. A simple indistinguishability argument shows that there is some finite round x, s.t. processes $p_4, \ldots, p_{k+2}$ have decided on $k-1$ distinct values: Just consider the execution where p_1 is a perpetual root among p_1, p_2, p_3 and $p_4, \ldots, p_n$ are as defined above perpetually. Since we assumed a correct algorithm, all processes $p_4, \ldots, p_{k+2}$ decide on some value by some round x. To see that there are $k-1$ different decisions, observe that $p_4, \ldots, p_{k+2}$ never learn a value that is not their own initial value, thus this fact follows from validity.

From round $x+1$ on, we use a communication graph $\mathcal{G}^r$ that is the same as above, except that the edge from p_1 to p_3 is removed and an edge from p_3 to p_4 is added (cf. Fig. 4.2b). Note carefully that the total number of root components is preserved, and that there are still no incoming edges to $\{p_1, p_2\}$. Since the resulting execution is admissible in $\mathcal{M}$, by a similar reasoning as above, there must be some round y s.t. p_3 decides by round y; its decision value must be in $\{x_1, x_2, x_3\}$, since p_3 can have heard at most from $\{p_1, p_2, p_3\}$. Obviously, p_1 and p_2 are still undecided.

Finally, it follows from the bivalent configuration of $\mathcal{A}_{|D}$ reached by round y, according to Lemmas 1 and 2, that there exist communication graphs Γ_1 resp. Γ_2 for all rounds $r > y$, which are the same as the graphs used for rounds $x < r \leq y$, except that the links between p_1 and p_2 are chosen such that they both decide on x_1 resp. x_2. We now continue our execution with Γ_2 if p_3 decided x_1, and with Γ_1 otherwise. Obviously, this guarantees that p_1, p_2 and p_3 reach at least two different decisions.

As we have now reached a total of $k+1$ decisions, we have established a contradiction. This completes the proof of Theorem 7. □

Theorem 7 reveals that an eventual globally stable interval after round r_{GST} is not enough to make k-set agreement possible. The reason for this is that already before r_{GST}, $(k+1)$ decisions could have been enforced. This is achieved by the possibility to hide decisions from subsequent root components via rearranging the root components in the system – in essence, nodes that have already decided cease to have outgoing edges and nodes that never even learned the proposal value on which this decision is based become roots that must eventually decide on a different value. In this scenario, the BRS-Theorem was not directly applicable (although the proof is similar in spirit), because property (C) of the BRS-Theorem cannot be established as the global stability round r_{GST} makes consensus solvable eventually.

CHAPTER 5

Predicated Dynamic Networks

5.1 The Predicate $\mathcal{P}_{\mathbf{srcs}}(k)$

In Chapter 4, we investigated the directed dynamic network (DDN) model, where communication is modeled by means of a sequence of round-by round communication graphs $\mathcal{G}^r$. The solvability/impossibility results in that chapter are mostly based on the restrictions on root components (cf. Definition 4), which are in a way the high-level structural properties of the graph. It seems worthwhile to investigate if we can find solvability boundaries for k-set agreement in the DDN model, which do not rely on such high-level assertions, but rather on some compact low-level predicate on the communication graphs. Ideally, we are looking for a low-level predicate from which we can derive tight impossibilities for k-set agreement. One such predicate, $\mathcal{P}_{\text{srcs}}(k)$, will be the focus of this chapter. It has been found in [9] to be sufficient for solving k-set agreement in the DDN model.

The predicate is based on the notion of a perpetually timely neighborhood of a process, which will allow us to define the stable skeleton of the communication graphs in a run, both of which we will now introduce (for a more detailed description, cf. [9, Section 2]). The round r *perpetually timely neighborhood* of a process p, $PT(p, r)$, is the set of those processes that p heard of, i.e. had an incoming edge from, in every round until r. The round r *stable skeleton* $G^{\cap r}$ of the communication graph $\mathcal{G}^r$ is a graph that contains those edges that were present in $\mathcal{G}^r$ in every round until r; those edges are called the *directed timely edges* of $\mathcal{G}^r$. The correspondence between $PT(p, r)$ and $G^{\cap r}$ is that a process q is in $PT(p, r)$ if and only if there is an edge $(q \to p)$ in $G^{\cap r}$ (for an example, cf. Fig. 5.1).

Going one step further, we define the stable skeleton for a whole execution, $G^{\cap\infty}$, as the intersection of all $G^{\cap r}$. In a similar fashion, we define the perpetually timely neighborhood of p, $PT(p)$, as the intersection of all $PT(p, r)$ of an execution:

Definition 5 (Perpetually Timely Neighborhood and Stable Skeleton from [9, equation (4)]). *Let $PT(p, r)$ contain those processes, process p received a message from in every round up until and including round r. Then, the perpetually timely neighborhood of p is defined as*

$$\begin{tikzcd}
& p_1 \arrow[dl] \arrow[dr] & \\
p_2 \arrow[rr] \arrow[dr] & & p_3 \arrow[ul, bend right] \\
& p_4 &
\end{tikzcd}
\qquad
\begin{tikzcd}
& p_1 \arrow[dl] \arrow[dr] & \\
p_2 \arrow[ur, bend left] \arrow[rr] \arrow[dr] & & p_3 \\
& p_4 &
\end{tikzcd}
\qquad
\begin{tikzcd}
& p_1 \arrow[dl] \arrow[dr] & \\
p_2 \arrow[ur, bend left] & & p_3 \arrow[ll] \arrow[ul, bend right] \arrow[dl] \\
& p_4 &
\end{tikzcd}$$

$\mathcal{G}^{\mathbf{1}}$ $\mathcal{G}^{\mathbf{2}}$ $\mathcal{G}^{\mathbf{3}}$

$$\begin{tikzcd}
& p_1 \arrow[dl] \arrow[dr] & \\
p_2 & & p_3 \\
& p_4 &
\end{tikzcd}$$

$\mathcal{G}^{\cap \mathbf{3}}$

Figure 5.1: The three communication graphs $\mathcal{G}^1, \mathcal{G}^2, \mathcal{G}^3$ have the round 3 stable skeleton $\mathcal{G}^{\cap 3}$. p_1 constitutes both $PT(p_2, 3)$ and $PT(p_3, 3)$, the round 3 perpetually timely neighborhood of p_2 and p_3, respectively. $PT(p_1, 3)$ and $PT(p_4, 3)$ are both empty, while $PT(p_3, 2)$ consists of p_1 and p_2.

$PT(p) := \bigcap_{r>0} PT(p, r),$
and the stable skeleton graph is
$\mathcal{G}^{\cap\infty} = \bigcap_{r>0} \mathcal{G}^r.$

While $PT(p)$ and $\mathcal{G}^{\cap\infty}$ are always well defined (possibly containing no edge), we will define predicates that ensure non-trivial stable skeletons. After all, we want to solve non-trivial agreement problems, which is much easier when the stable skeleton is sufficiently well connected.

The predicate $\mathcal{P}_{\text{srcs}}(k)$ states that, in every round of an execution, every set of $k+1$ processes in the communication graph contains at least two fixed processes $q \neq q'$ that receive a message from the same fixed process p. We call p a *two-source*, as it is a source of messages for at least two processes. Since we assume that every process receives its own message, it is possible that $q = p$ or $q' = p$ (cf. the example depicted in Fig. 5.2).

For the formal definition of $\mathcal{P}_{\text{srcs}}(k)$ we first introduce the predicate $\mathcal{P}_{\text{src}}(p, S)$. It states that p is a perpetual two-source for two nodes in S, i.e. p is a two-source for the same two nodes $q \neq q'$ in every round, by stating that p is in their perpetually timely neighborhood $PT(.)$:

Definition 6 ([9, equation 8, line 1])**.**
$\mathcal{P}_{src}(p, S) :: \exists q, q' \in S, q \neq q' : p \in (PT(q) \cap PT(q'))$

Using this definition we can express $\mathcal{P}_{\text{srcs}}(k)$ by stating that $\mathcal{P}_{\text{srcs}}(p, S)$ holds for any set S of size $k + 1$.

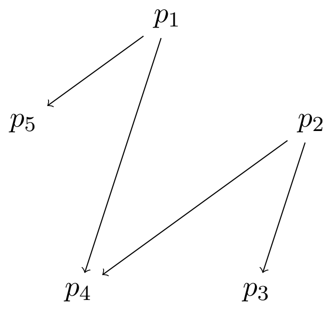


Figure 5.2: This graph satisfies $\mathcal{P}_{\text{srcs}}(2)$ (and hence $\mathcal{P}_{\text{srcs}}(i)$ for all $i \geq 2$): in every set of 3 or more nodes there are two nodes with an incoming edge from the same node (every node is considered to have an edge from itself to itself). However it does not satisfy $\mathcal{P}_{\text{srcs}}(1)$: e.g. p_3 and p_5 have no incoming edge from a common node.

Definition 7 ([9, equation 8, line 2]).
$\mathcal{P}_{srcs}(k) :: \forall S, |S| = k+1\ \exists p \in \Pi : \mathcal{P}_{src}(p, S)$

We might ask ourselves how precisely $\mathcal{P}_{\text{srcs}}(k)$ makes k-set agreement possible, and identify two important aspects: The first is that we defined $\mathcal{P}_{\text{srcs}}(k)$ using the perpetually timely neighborhoods. This ensures that links that are missing at some point are not considered by the predicate at all, which essentially implies that at least the stable skeleton satisfies the property of $\mathcal{P}_{\text{srcs}}$ forever. Such a perpetually stable skeleton alleviates the issues that arise when partitions may withhold information combined with the ability of components to restructure, by merging old root components and introducing new root components, as discussed in Section 4.4.

The second important aspect is that $\mathcal{P}_{\text{srcs}}(k)$ imposes limitations on the amount of root components present in the system: Because $\mathcal{P}_{\text{srcs}}(k)$ asserts that in every set of $k+1$ processes, two processes received a message from another process, this implies that there cannot be $k+1$ or more disjoint root components in $\mathcal{G}^{\cap\infty}$ (cf. [9, Theorem 1]).
Although the predicate $\mathcal{P}_{\text{srcs}}(k)$ is far from being the weakest predicate that allows k-set agreement to be solved, it is tight, in the sense that $(k-1)$-set agreement is impossible when $\mathcal{P}_{\text{srcs}}(k)$ is the only assertion about the system. When investigating this $(k-1)$-set agreement impossibility, we find that $\mathcal{P}_{\text{srcs}}(k)$ allows us to create a set L of $(k-1)$ isolated processes and a set of processes where every process receives messages from some process $s \notin L$. When starting with distinct initial values, the $(k-1)$ processes in L and s will, by termination, decide on their own values eventually, leading to k decision values (cf. [9, Theorem 2]).

5.2 Impossibility of k-set agreement with Moving Source

We now ask ourselves how we could weaken the predicate $\mathcal{P}_{\text{srcs}}(k)$ while still maintaining the solvability of k-set agreement. A promising idea might be to use a weaker predicate that ensures that every set of $k+1$ processes contains at least two processes q, q', which receive a message from the same process p_r in every round r. Accordingly, we define the round r predicate $\mathcal{P}^r_{\text{srcs}}(k)$:

Definition 8 (Round r variant $\mathcal{P}^r_{\text{srcs}}(k)$ of $\mathcal{P}_{\text{srcs}}(k)$)**.**
$\mathcal{P}^r_{src}(p, S) :: \exists q, q' \in S, q \neq q' : (p \to q) \in \mathcal{G}^r \wedge (p \to q') \in \mathcal{G}^r$
$\mathcal{P}^r_{srcs}(k) :: \forall S, |S| = k + 1\ \exists p_r \in \Pi : \mathcal{P}^r_{src}(p_r, S)$

Note that, as in the original $\mathcal{P}_{\text{srcs}}(k)$ predicate, we assume that $\forall p \in \mathcal{G}^r : (p \to p) \in \mathcal{G}^r$ and $p = q$ or $p = q'$ is allowed.

We now show that this predicate is too weak, i.e., that k-set agreement is impossible with only $\mathcal{P}^r_{\text{srcs}}(k)$ by applying the BRS-Theorem (Theorem 1).

Theorem 8. *Solving k-set agreement is impossible in the two-source model with $n > k$ if the only restriction is that $\mathcal{P}^r_{srcs}(k)$ holds for all rounds r.*

Proof. By contradiction. Suppose there exists an algorithm $\mathcal{A}$ that solves k-set agreement under these conditions. We show that there exists a communication graph sequence such that $\mathcal{A}$ satisfies the conditions of the BRS-Theorem. Note that $n > k$ enables us to make the following partitioning:

Fix D_i in such a way that $|D_i| \geq 1$ for $0 < i < k$ and fix $\overline{D}$ in such a way that $|\overline{D}| = 2$.
We choose the communication graph in each round such that the processes of D_i, $0 < i < k$, are strongly connected and the processes in $\overline{D}$ are weakly connected, while there is no connection between the processes of any two partitions.

First, we establish that this partitioning indeed satisfies the predicate we are interested in in the following lemma:

Lemma 3. *In the above partitioning, $\mathcal{P}^r_{srcs}(k)$ holds in all rounds r.*

Proof. Fix any set S of size $k + 1$. We show that for S and some p_r, $\mathcal{P}^r_{\text{src}}(p_r, S)$ holds. Note that because there are k partitions, at least two distinct processes $q, q' \in S$ must be from the same partition.
If $q, q' \in D_i$, $0 < i < k$, then by the strong connectedness of D_i, $(q \to q') \in \mathcal{G}^r$ and therefore $\mathcal{P}^r_{\text{src}}(q, S)$ holds.
If, on the other hand, $q, q' \in \overline{D}$, by the weak connectedness of $\overline{D}$ and because $\overline{D} = 2$, we have that $(q \to q') \in \mathcal{G}^r \vee (q' \to q) \in \mathcal{G}^r$ holds, which implies that $\mathcal{P}^r_{\text{src}}(q, S) \vee \mathcal{P}^r_{\text{src}}(q', S)$ holds. $\square$

We can now finish the proof by showing that this partitioning satisfies the conditions of the BRS-Theorem.

(A) $\mathcal{R}_{(\overline{D})}$ is nonempty:
Note that we have defined $\mathcal{G}^r$ such that, for all rounds r, it holds that there are no messages sent to a process of $\overline{D}$ by a process of some D_i, $0 < i < k$. By Lemma 3, these communication graphs correspond to an admissible execution. Therefore, by the assumed correctness of $\mathcal{A}$, the nodes in $\overline{D}$ must eventually decide without receiving any messages from any process not in $\overline{D}$.

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$:

Note that in addition to the facts mentioned in (A), our communication graphs ensure that there is no communication between any two partitions D_i and D_j, $0 < i \neq j < k$.

Pick any run $\rho \in \mathcal{R}_{(\overline{D})}$. Create a new run ρ', which is the same as ρ except that the nodes in D_i, $0 < i < k$, start with unique initial values in ρ'. Because there are no messages exchanged between any two partitions, by the assumed correctness of $\mathcal{A}$, the processes of D_i must eventually decide on a proposal value from D_i, hence establishing $\rho' \in \mathcal{R}_{(D,\overline{D})}$. Also the nodes of $\overline{D}$ in ρ' make exactly the same state transitions as in ρ, because the initial state of the nodes in $\overline{D}$ as well as the sequence of communication graphs is the same in both runs, establishing $\rho \overset{\overline{D}}{\sim} \rho'$.

(C) Consensus is impossible in $\mathcal{M}' = \langle \overline{D} \rangle$:

Because $|\overline{D}| = 2$ and $\overline{D}$ is weakly connected, we may construct a communication graph for each round in such a way that the two processes q, q' of $\overline{D}$ are either connected by a link $(p \to q)$ or by a link $(q \to p)$ or by a link $(p \leftrightarrow q)$. By Theorem 3, consensus is impossible in such a setting.

(D) $\mathcal{M}'_{\mathcal{A}_{|\overline{D}|}} \preccurlyeq_{\overline{D}} \mathcal{M}_{\mathcal{A}}$:

Fix any run $\rho' \in \mathcal{M}'_{\mathcal{A}_{|\overline{D}|}}$ and consider the run $\rho \in \mathcal{M}_{\mathcal{A}}$ where every process in $\overline{D}$ has the same initial state and the same sequence of communication graphs in ρ as in ρ'. Since in our sequence of communication graphs messages are never sent between any two partitions anyway, we immediately have $\rho \overset{\overline{D}}{\sim} \rho'$.

$\square$

Note how the key argument in this proof was again the partitionability of the system, as per our round-by-round communication graph model, and the impossibility of consensus in one partition $\overline{D}$. The construction we used to make consensus impossible in $\overline{D}$ was that there is a *moving two-source* in $\overline{D}$, i.e. a two-source which may move in every round.

CHAPTER 6

General Omission Failures

In the DDN model of Chapters 4 and 5, there is no notion of a faulty process. Rather, following the view introduced in [43], every process may suffer from communication failures on its incoming and/or outgoing links. In this chapter, we will take the more classic view, which attributes communication failures to the sending or receiving process.

6.1 Omission Failures

We already introduced crash failures, where processes may stop operating at any point, in Section 2.3. Omission failures [39, 41] are a more general failure model that include crash failures as a special case. We consider only synchronous systems here and define three types of omission failures:

1. A *send omission* failure is said to occur at a process, if it sends messages only to a subset of processes, in some round. Note that this also covers crash failures.
2. Similarly, a process commits a *receive omission* failure, if it receives only a subset of the messages that were sent to it or crashes.
3. A *general omission failure* is said to occur when a process may commit send omissions as well as receive omissions.

When modeling the sending of messages at a process as putting the messages into a remote buffer, respectively the receiving of messages as reading the messages from a local buffer, an omission failure corresponds to a faulty buffer. For example, a local buffer overflow at some process could be modeled as a receive omission failure of this process.

By their definition, omission failures include crash failures as special cases. This is justified by the fact that a crashed process behaves the same with respect to the sending and receiving of messages as a process that omits all processes from some round on. We can therefore conclude that an algorithm that is resilient to t omission failures is also resilient to t crash failures [41].

Analogously, when we have an impossibility result for crash failures, this impossibility holds also for omission failures. However, note carefully that omission failures are a more severe failure mode than crash failures, as e.g. a process that did not send to some other process in a round may do so in a later round. This means that problems that are solvable in the crash failure model are not necessarily solvable in the general omission failure model.

6.2 Round Complexity

One way of evaluating the performance of a synchronous algorithm is to determine its round complexity, i.e., the time it takes the algorithm to reach a terminal configuration. When developing algorithms one ideally looks for optimal round complexity, that is, algorithms with a worst-case round complexity just above the lower bound round complexity for which there exists an impossibility proof. If no such optimal algorithm can be found, this may be an indication that the lower bound is not tight [26].

A classic round complexity result is that t rounds are insufficient to solve consensus in the presence of t crash failures [24]. This can be established by a bivalency proof [2], where it is shown that every consensus algorithm has a run where it stays bivalent for $t-1$ rounds and undecided for one additional round. Informally, this proof inductively shows that in every round $\leq t-1$ we can find a single crucial process p that determines the valency of the configuration in some failure-sparse execution, i.e. an execution where there is a single crash failure in every round. The proof is then concluded by extending the execution for one additional round and showing that not all processes can have decided at the configuration reached.

As the worst-case round complexity often occurs only in a few "exotic" executions, it is advantageous to develop so-called early stopping algorithms that can terminate before the worst-case round complexity in favorable executions [38]. In case of early stopping consensus [23], for example, every deterministic algorithm needs at least $\mathit{min}(t+1, f+2)$ rounds in executions where $f \leq t$ processes crash.

By the arguments provided in the previous section, we can conclude that the round complexity result for consensus with t crash failures holds also for omission failures:

Theorem 9. *There is no algorithm that solves consensus in the presence of* t *general omission failures in* $\leq t$ *rounds.*

6.3 k-Set Agreement with Omission Failures

A tight bound for the solvability of k-set agreement in synchronous systems with omission failures is provided in [41, Theorem 2] as:

$$t < \frac{k}{k+1}n \tag{6.1}$$

The tightness of this bound means that k-set agreement is solvable if and only if less than $\frac{k}{k+1}n$ send omission failures occur in a run: there is an impossibility proof, based on a classic partitioning argument, for $t \geq \frac{k}{k+1}n$, and an algorithm for $t < \frac{k}{k+1}n$, as presented in [41, Figure 5].

The algorithm from [41, Figure 5] works as follows: Every process p sends its proposal values to all other processes in every round. In addition, p stores locally a list of trusted processes that corresponds to those processes that p received a message from. If p does not receive a message from another process q in some round, p removes q from its list. Only if p has at least $n - t$ processes in its list after $t - k + 2$ rounds, will it decide on the minimum of all received proposal values.

Using a complex correctness proof, it is shown that this algorithm solves k-set agreement with $t < \frac{k}{k+1}n$ general omission failures in $t - k + 2$ rounds. It is an open question whether this round complexity is optimal (cf. [41, Section 6.3]).

In an attempt to possibly answer this question, we will apply the BRS-Theorem (Theorem 1) in this setting. Note that general omission failures essentially empower the adversary to let faulty processes receive only a subset of the transmitted messages. The adversary can hence simply let the processes of a single partition D_i send only messages to and receive only messages from processes within the very same partition, thereby immediately establishing property (D) of the BRS-Theorem. Moreover it tells us that properties (A) and (B) of the BRS-Theorem can be easily established if isolated decisions within the partitions can be enforced, as discussed subsequently.

First, let us consider $|D_i|$, the size of the partitions D_i for $0 < i < k$. In order to enforce a decision within a partition D_i, the processes in that partition must not be able to determine that they are faulty. If they could reliably diagnose their own fault, an algorithm could simply not let them decide since the *termination* condition states that only correct processes must decide. This leads to the conclusion that we need as many faulty processes in a partition as there are correct processes in the system, i.e. $n - t$. Then, each processes in D_i receives the same number of messages it would if it were correct and therefore has no way of determining that it is in fact faulty.

Second, let us look at $|\overline{D}|$, the size of the partition $\overline{D}$. Since $\overline{D}$ is the last remaining partition and the previously discussed partitions contained only faulty processed, $\overline{D}$ contains at least all correct processes, thereby easily establishing property (A) of the BRS-theorem. Furthermore, recall that, for the BRS-Theorem to establish an impossibility for k-set agreement, we need a corresponding impossibility for consensus in $\overline{D}$, which corresponds to property (C) of the theorem. Therefore, for an impossibility of k-set agreement in x rounds, we would need x general omission failures in $\overline{D}$. Looking at the failure bound from Eq. (6.1), we could take an "educated guess" at the number of failures remaining in $\overline{D}$ and conjecture that they are in the order of $\frac{t}{k}$. This can be justified formally by applying the BRS-Theorem to the general omission failure model.

Theorem 10. *There exists no k-set agreement algorithm that terminates before $\frac{t}{k}$ rounds in the general omission failure setting.*

Proof. For the purpose of deriving a contradiction, assume that some algorithm $\mathcal{A}$ solves k-set agreement in the given model in at most $x \leq \frac{t}{n}$ rounds. We show that this implies the existence of a partitioning such that the conditions of Theorem 1 are satisfied.

For D_i we use partitions of size $n - t$. We define them employing the syntax from [8, Theorem 2]: Let $\ell = n - t$; for $1 \le i < k$, define $D_i = \{p_{(i-1)\ell+1}, \dots, p_{i\ell}\}$ such that $|D_i| = \ell$, and let $D = \bigcup_{1 \le i \le k-1} D_i$. From Eq. (6.1), we obtain $\frac{t}{k} < \frac{n}{k+1}$ and $n - t > n\left(1 - \frac{k}{k+1}\right) = \frac{n}{k+1}$, which yields

$$n - t > t/k. \tag{6.2}$$

For x, the number of remaining faulty processes in $\overline{D}$, we obtain

$$x = t - (n-t)(k-1) < t\left(1 - \frac{k-1}{k}\right) = \frac{t}{k} \tag{6.3}$$

which implies the admissibility of the runs in our partitioning where $|D| + \frac{t}{k} - 1$ (resp. $|D| + \lfloor\frac{t}{k}\rfloor$) processes are omission faulty, in the case of $\frac{t}{k} \in \mathbb{N}$ (resp. $\frac{t}{k} \notin \mathbb{N}$).

We conclude the proof by showing that this partitioning satisfies conditions (A)-(D) of Theorem 1.

(A) $\mathcal{R}_{(\overline{D})}$ is nonempty:
Consider the runs where the processes of D omit sending messages to any process of $\overline{D}$. Since our partitioning allows $|D|$ general omission faults, these runs are admissible. Moreover, as the $n - t$ correct processes of $\overline{D}$ send their messages to every other correct process of $\overline{D}$, because $\mathcal{A}$ solves k-set agreement with the given failure bound, these correct processes eventually decide. This establishes $\mathcal{R}_{(\overline{D})} \neq \emptyset$.

(B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$:
Consider the set of runs $\mathcal{H}$ where processes of D_i start with input value $x_i = i$, $1 \le i \le k-1$. Moreover, for $1 \le i \le k-1$, let the processes of D_i omit sending and receiving messages to, resp. from, any process of $\Pi \setminus D_i$. Again, since our partitioning allows $|D|$ general omission faults, $\mathcal{H} \neq \emptyset$. Also, since processes of D_i receive messages from $n - t$ other processes (of D_i), they will eventually decide on x_i because $\mathcal{A}$ solves k-set agreement with the given failure bound.
Pick any ρ' from $\mathcal{R}_{(\overline{D})}$. Note that the processes of $\overline{D}$ receive no messages until they decide in both $\mathcal{R}_{(\overline{D})}$ and $\mathcal{H}$ and the proposal values of the processes of $\overline{D}$ are not restricted in $\mathcal{H}$. Thus, we can find a ρ in $\mathcal{H}$ with $\rho \overset{\overline{D}}{\sim} \rho'$. Since $\mathcal{H} \subseteq \mathcal{R}_{(D,\overline{D})}$ we have established $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$.

(C) Consensus is impossible in $\mathcal{M}' = \langle \overline{\mathbf{D}} \rangle$:
Consider a system $\mathcal{M}' = \langle \overline{D} \rangle$ that has the same system assumptions as $\mathcal{M}$ with the restriction that $x < \frac{t}{k}$ processes may be omission faulty in $\mathcal{M}'$. This may occur in every run of $\mathcal{A}$, since our partitioning in combination with the failure bound allows up to x omission faults. By Theorem 9, consensus is impossible in $\mathcal{M}'$ in x rounds.

(D) $\mathcal{M}'_{\mathcal{A}_{|\overline{D}}} \preccurlyeq_{\overline{D}} \mathcal{M}_{\mathcal{A}}$:
Fix a run $\rho' \in \mathcal{M}'_{\mathcal{A}_{|\overline{D}}}$. Take a run ρ from $\mathcal{M}_{\mathcal{A}}$, where the processes of D never send any message to a process of $\overline{D}$ and the processes of $\overline{D}$ in $\mathcal{A}$ make the same state transitions as in ρ'. ρ exists because our partitioning allows $|D|$ omission faults and $\mathcal{A}_{|\overline{D}}$ is a restriction of $\mathcal{A}$ according to Definition 1.

□

Interestingly, there exists a k-set agreement algorithm that terminates in $\lfloor \frac{t}{k} \rfloor + 1$ rounds and tolerates up to t *crash failures* and a corresponding impossibility for $\lfloor \frac{t}{k} \rfloor$ rounds, based on topological arguments [18]. For $\frac{t}{k} \notin \mathbb{N}$, we have $\lfloor \frac{t}{k} \rfloor < \frac{t}{k}$ and hence, for these values, our bound for omission failures matches exactly this bound for crash failures. Thus, we have to conclude that, unfortunately, the BRS-Theorem does not allow us to determine whether general omission failures increase the round complexity when compared to crash failures: The lower bound proof from [18] shows that this bound holds even for the less severe crash failures (albeit its proof requires topological arguments). Note carefully that for models such as the crash failure model, where a complete partitioning of the processes cannot be established, the BRS-Theorem is not directly applicable. Very different arguments equivalent to the ones that can be found in the according topological proofs are required here, which are rooted in a very different source of impossibilities than partitioning, namely, some generalized "symmetry breaking" impossibility. We will make a very similar observation regarding the shared memory model in Chapter 7, where we will discuss this issue in more detail.

From the previous explanations, we can also conclude that for algorithms with *strong termination* (also called uniform algorithms [17]), where all non-crashed faulty processes must also decide (in particular, omission faulty processes that continue to make steps), the BRS-Theorem immediately provides us with the lower bound of $t - k + 3$ rounds: We choose for D_i with $0 < i < k$ single-process partitions of omission faulty processes that neither send nor receive any messages but never cease to make steps. Because of strong termination, there are executions where these processes decide on $k - 1$ distinct values. This leaves $t - (k - 1) = t - k + 1$ faulty processes for $\overline{D}$, which corresponds to the impossibility of consensus in $\overline{D}$ in $t - k + 2$ or fewer rounds, as the round complexity lower bound for uniform consensus with f crash failures is $f + 2$ rounds [17].

CHAPTER 7

Shared Memory

In this final chapter, we attempt to adapt the BRS-Theorem from Chapter 3 to the shared memory model of computation from Section 2.2. We show that certain parts of the theorem can be carried over literally to this fundamentally different model, and elaborate on the only part in for which we could not yet find a suitable counterpart. We conclude the chapter with a detailed discussion of the non-topological impossibility result for k-set agreement in shared memory by Attiya and Castañeda [4, 5] and compare the approaches taken therein to the BRS-theorem.

7.1 Immediate Snapshot Executions

We already briefly introduced immediate snapshot (IS) executions in Section 2.2. Subsequently we will see that they provide a very useful abstraction in the sense that they greatly reduce the complexity of analyzing all the different interleaving access schemes to the shared memory. Because we will require more detailed insights about IS-executions in this chapter, we will now proceed to examine them in more detail. For simplicity, we will assume that the considered IS executions are *minimal final*, i.e. processes don't take any steps after terminating.

Consider the exemplary IS execution $\varepsilon = \{p_1, p_2\}\,\{p_3\}$, where p_1 and p_2 concurrently write to the shared memory (and subsequently both read from it), followed by p_3 writing to (and reading from) the shared memory. Observe that in systems where shared memory is the only means of exchanging information between processes, ε prohibits p_1 and p_2 from "seeing", i.e. learning anything about, p_3. In general, in executions of the form $\varepsilon\,\{p_i\}^r$ with $r \geq 1$, $\varepsilon \neq \emptyset$ and $p_i \notin \varepsilon$, we thus say that that p_i is *unseen* in the execution. Accordingly, we say a process is *seen* in an execution when it is not unseen in the execution.

Even though they are a significant restriction on the possible access schemes of the shared memory, immediate snapshot executions still contain uncertainty. We will summarize here briefly two main statements about this uncertainty. A comprehensive study of IS executions, which includes detailed proofs of the two statements presented here as well as an elaboration on topological aspects of IS executions, can be found in [6]. We will make use of their notation $\varepsilon \overset{\neg p}{\sim} \varepsilon'$ to express that ε and ε' are indistinguishable for all processes of $\Pi \setminus \{p\}$.

In order to gain some insight into the uncertainty of IS executions, let us investigate the following issue: Considering the above example execution $\varepsilon = \{p_1, p_2\}\,\{p_3\}$, can we find an execution $\varepsilon' \neq \varepsilon$ that is indistinguishable from ε to p_2 and p_3 but not to p_1, i.e. find ε' s.t. $\varepsilon \overset{\neg p_1}{\sim} \varepsilon'$ holds?

To answer this question, we could first look closely at what p_3 observes in ε. Obviously, p_3 sees what p_1 and p_2 wrote to their respective single-writer registers. But can p_3 deduce in which order p_1 and p_2 wrote to the shared memory? This answer can be reduced to determining whether p_1 or p_2 can observe in which order they make their step *and* whether they can tell anybody about it. The crucial thing to note here is that while p_1 and p_2 both see the other, they do so only *after* writing to the shared memory since they take their snapshot of the memory after they wrote to it (and thereby render themselves visible). Thus, although once p_3 takes its step, it could deduce the ordering of the steps taken by p_1 and p_2 from their combined local views, since those views have been obtained only after p_1 and p_2 wrote to the shared memory, p_3 has no way to learn them.

This means that p_3 cannot distinguish between $\varepsilon = \{p_1, p_2\}\,\{p_3\}$, $\varepsilon_1 = \{p_1\}\,\{p_2\}\,\{p_3\}$, and $\varepsilon_2 = \{p_2\}\,\{p_1\}\,\{p_3\}$. Since we were interested in some ε' that is distinguishable from ε only for p_1, we can choose ε_1 as the desired ε': Because p_2 cannot determine locally whether p_1 read from the shared memory just before p_2 wrote to it or concurrently with p_2, we have $\varepsilon \overset{\neg p_1}{\sim} \varepsilon_1$. Regarding ε_2, note carefully that $\varepsilon \overset{\neg p_2}{\sim} \varepsilon_2$ holds.

It is intuitively clear that we could not have chosen any other distinct ε', even in the more general case of an execution that contains many more steps and is of the type
$\alpha = s_1 \dots s_l\,\{p_1, p_2\}\,\{p_3\}\,s_{l'} \dots s_{l''}\,\{p_1\}^t$ where $t \geq 0$ and the step of p_1 after s_l is its last observable step (formally $\forall m \in [l', l''] : p_1 \notin s_m$):
Here, for $\alpha' = s_1 \dots s_l\,\{p_1\}\,\{p_2\}\,\{p_3\}\,s_{l'} \dots s_{l''}\,\{p_1\}^{t'}$ with some unique $t' \geq 0$ that is chosen so that p_1 terminates in α', we have that α' uniquely satisfies $\alpha \overset{\neg p_1}{\sim} \alpha'$; if putting p_1 even further into the prefix of α resulted in a different execution, then some process other than p_1 would notice that and distinguish between the executions. Moreover, if we modified the view of p_1 in α in any step but its last observable step, p_1 could make this different view known to the other processes (at latest) in its last observable step.

Accordingly, if we are given ε' as above and want to find $\varepsilon \neq \varepsilon'$ s.t. $\varepsilon \overset{\neg p_1}{\sim} \varepsilon'$, it is clear that there is only one possible choice for ε (of course this is also true for α' and α).

Finally, let us investigate what would happen if we wanted to find an execution ε'' that is distinct from ε but $\varepsilon \overset{\neg p_3}{\sim} \varepsilon''$ holds. Obviously, it is not really possible to find such an ε'', because as soon as we move $\{p_3\}$ to a different location in ε some process will "see" p_3 and the execution will no longer be indistinguishable for this process. Additionally, as we restricted ourselves to minimal final executions, appending $\{p_3\}^t$ for some $t > 0$ to ε does not constitute an admissible execution. The reason for this is that p_3 is unseen in ε. Generally speaking, when we have an execution of the form $\alpha = s_1 \dots s_l\,\{p\}^t$ with $t > 0$ and $\forall m \in [1, l] : p \notin s_m$, for the same reasons as above, it is clear that no execution α'' can exist for which $\alpha \overset{\neg p}{\sim} \alpha''$ holds.

In summary, we have the following two lemmas that tell us about the uncertainty of IS-executions. They are in essence Lemma 3.3 and Lemma 3.4 from [6], resp. Lemma 1 and Lemma 2 from [4], where formal and detailed proofs of their correctness can be found.

Lemma 4. *If a process p is seen in an IS execution α, then there exists a unique IS execution $\alpha' \neq \alpha$ such that $\alpha \overset{\neg p}{\sim} \alpha'$ holds.*

Lemma 5. *If a process p is unseen in an IS execution α, then there is no IS execution $\alpha' \neq \alpha$ s.t. $\alpha \overset{\neg p}{\sim} \alpha'$ holds.*

7.2 Applying the BRS-Theorem to Shared Memory

We will now proceed by explaining how properties (A), (C) and (D) of the BRS-Theorem (Theorem 1) can readily be applied to the shared memory model and elaborate what the issues of applying property (B) are.

Runs exist where the processes of $\overline{D}$ decide before hearing from D, i.e., (A) $\mathcal{R}_{(\overline{D})} \neq \emptyset$

While Theorem 1 states that $\mathcal{R}_{(\overline{D})}$ describes the runs where the processes in $\overline{D}$ decide before receiving a message from the processes in D, the purpose of this property is to ensure that no process in $\overline{D}$ knows an initial value of any process in D. This is straightforward to establish in shared memory with asynchronous processes if there may be $|D|$ failures, i.e., depending on the actual partitioning, the failure bound f must at least be greater or equal to $k-1$ (as each of the $k-1$ partitions D_i must at least contain one process): Consider for instance a run ρ where the processes in D crash before making a step. Obviously, the remaining processes in ρ, namely the processes in $\overline{D}$ will eventually decide without ever learning a proposal value of any process in D, establishing $\mathcal{R}_{(\overline{D})} \neq \emptyset$.

An alternative to initially dead processes in D would be to consider runs where the processes in $\overline{D}$ decide before a process in D makes a step. Again, such runs are admissible if at least $f \geq |D|$ may crash, as then the processes in $\overline{D}$ cannot distinguish whether the processes in D are just scheduled late or have crashed.

In all runs it holds that (C) Consensus is impossible in $\mathcal{M}' = \langle \overline{D} \rangle$

This may be even more straightforward to establish than the previous result, as it can be achieved simply by allowing a single crash failure to occur in $\overline{D}$, irrespectively of the failure pattern of the run. We can accomplish this by setting the failure bound $f > |D|$, which means that the failure bound must be set at least to $f \geq k$. The impossibility of consensus in $\mathcal{M}'$ can subsequently be derived directly from the result of [7, Theorem 5.24]. Note that this result shows that if we want to prove the impossibility of k-set agreement for $f \geq k$, as it is established in the topological proofs from [14, 19, 32], we had indeed to choose a partitioning where $|D| \leq k-1$. Since we need $k-1$ decisions in D, in order to apply the BRS-Theorem, this implies that for this tight impossibility bound we need $|D| = k-1$, i.e. D consists of $k-1$ single-process partitions.

The runs of $\mathcal{M}'_{A_{\overline{D}}}$ can be observed in $\mathcal{M}$, i.e. (D) $\mathcal{M}'_{\mathbf{A}_{|\overline{\mathbf{D}}}} \preccurlyeq_{\overline{D}} \mathcal{M}_A$

Recall that the restriction of an algorithm $A_{|\overline{D}}$ is defined with respect to sending messages (cf. Definition 1). Since we only consider single-writer, multi-reader registers for shared memory here, when we want to run A on the restricted system $\mathcal{M}' = \langle \overline{D} \rangle$, we don't need to make any additional modifications to A. That is, for our purposes in this chapter, $A_{\overline{D}} = A$.

In order to establish the compatibility itself, we simply have to consider those runs $\mathcal{E} \subseteq \mathcal{M}_A$ where the processes not in $\overline{D}$ are initially dead. To ensure $\mathcal{E} \neq \emptyset$, i.e., guarantee that the runs just described are admissible, we again need the failure bound $f \geq |D|$, resp. $f \geq k-1$. For an arbitrary run ρ of $\mathcal{M}'_{A_{\overline{D}}}$ it is then easy to find a run ρ' in $\mathcal{E}$ where the processes of $\overline{D}$ make the same state transitions in ρ and ρ'. This establishes $\rho \overset{\overline{D}}{\sim} \rho'$ and thereby shows the desired compatibility.

The remaining problem – establishing (B) $\mathcal{R}_{(\overline{D})} \preccurlyeq_{\overline{D}} \mathcal{R}_{(D,\overline{D})}$

Undoubtedly the hardest question when we try to apply the BRS-Theorem to shared memory is how to establish the compatibility between the runs where the processes of $\overline{D}$ decide before reading the contents of the single writer registers of D with the runs where the processes of $\overline{D}$ behave in the same way but, additionally, the processes of D decide on $k-1$ distinct values that were initial values of processes in D.

The central task is finding out the precise nature of the runs $\mathcal{R}_{(D,\overline{D})}$. At first, it might seem that we could avoid this by glossing over the details of $\mathcal{R}_{(D,\overline{D})}$, since it is not directly required to describe them but merely to prove that they are compatible, wrt. $\overline{D}$, to $\mathcal{R}_{(\overline{D})}$. However, in order to do so, we need to show that the properties of some $\rho' \in \mathcal{R}_{(\overline{D})}$ do not impair the ability to form a run $\rho \in \mathcal{R}_{(D,\overline{D})}$ with $\rho \overset{\overline{D}}{\sim} \rho'$. To accomplish this, we most certainly need to have at least some insight into the inner workings of $\mathcal{R}_{(D,\overline{D})}$.

One idea would be to employ our knowledge about IS executions, previously introduced in this chapter. They indeed give us a hint about a direct impossibility of k-set agreement: Suppose a very simplistic algorithm uses some very straightforward basis like the largest known value to determine whether a value is "better" suited as a decision than another one. In the wait-free setting, i.e., with up to $f = n-1$ failures, the run where every p_i starts with input value $x_i = i$ and only a single process takes steps until it decides is admissible. There, obviously, the execution $\{p_1\}^{z_1} \{p_2\}^{z_2} \ldots \{p_n\}^{z_n}$, where z_i corresponds to the number of steps it takes p_i to decide at the respective position in the execution[1], will lead to n decision values. However this result is only of very limited immediate use, as it severely restricts the class of algorithms that we may consider – besides that it is not at all clear whether something like a "better" decision value exists in general.

A major complication apparently comes from the fact that as a process p of D takes a step, the processes of $\overline{D}$ are seen by p and p can most likely deduce their decision. Another idea would hence be to restrict the meaning of communicating information from p to q to *decision*

[1]Note that the existence of z_i is guaranteed by the assumed wait-freedom: A process p can never wait for a process $q \neq p$ to take a step as q might have crashed.

values: Even though q may observe the state of p via the shared memory, this does not mean that it can (always) infer p's decision value before p decides. We could thus try and exploit the impossibility of consensus with a single crash failure [34, Section 12].

More specifically, we know that if we allow a process failure in $D_1 \cup D_2$, then consensus is impossible in $D_1 \cup D_2$. Therefore, if we let the processes of D_i start with initial value $x_i = i$, and, for $1 \leq i < k$, we let a single process fail in every set $D_i \cup D_{i+1}$, then these sets could not solve consensus pairwise among themselves. While this may sound promising at first sight, there is an immediate problem with this approach:
The impossibility of consensus only means that there exists (possibly only) one execution where consensus fails in $D_i \cup D_{i+1}$. This means that when we investigate $\mathcal{R}_{(D)}$ for $k \leq 3$, a run where consensus is impossible among D is guaranteed to exist. For $k > 3$ things become more complex: Here, we actually would need to show for all[2] i with $1 \leq i \leq \frac{k-1}{4}$ that one of the executions $\varepsilon_{i,i+1}$ where consensus fails in $\mathcal{M}_{i,i+1} = \langle D_i \cup D_{i+1} \rangle$ is indistinguishable for the processes of D from the execution $\varepsilon_{i+2,i+3}$ where consensus fails in $\mathcal{M}_{i+2,i+3} = \langle D_{i+2} \cup D_{i+3} \rangle$. Unfortunately we could not find an immediate argument for why it should be possible to establish this indistinguishability. In fact, given that such an argument very much resembles the higher-order indistinguishability of configurations used in topological proofs [14, 19, 32], this is not surprising.

Finally, an alternative way to overcome this problem could be to generalize bivalence to k-valence and to try to show that starting from a $k+1$-valent initial configuration, we can find for every $k+1$-valent configuration, a $k+1$-valent successor configuration. This would need an invariant such as in every $k+1$-valent configuration, there exists a process p s.t. when p takes a step, the valency of the configuration is not reduced. In a way, such a process p would correspond to a non-critical process from [7, Lemma 5.17]. Actually, approaches such as this one have been tried in the past and lead to the topological impossibility proofs of k-set agreement from [14, 19, 32].

The problems that arise when adopting the BRS-Theorem to shared memory systems could hence be summarized as the following fundamental conflict: The key arguments of the BRS-Theorem require us to make a complete partitioning with respect to messages sent and received. In the shared memory model, this would correspond to a partitioning with respect to sharing some *certain* knowledge. Actually, we already faced a similar problem in the previous chapter when considering crash failures and general omission failures: Only the latter allowed us to establish a completely partitioned system without the processes reliably diagnosing (own) faults. Interestingly, the impossibility of achieving certain knowledge is fundamentally different from the impossibility of conveying information as used in the partitioning argument in the BRS-Theorem. As exploited in the topological proofs [14, 19, 32], the former is the perpetuated discrepancy in the view of all the processes, as delivered, e.g., by the application of Sperner's Lemma, that does not allow any pair processes in the system to ever share certain knowledge.

[2] For ease of notation during this simple illustration let us assume $k \equiv 1 \bmod 4$.

7.3 A Non-topological Impossibility Using a Graph of Executions

The impossibility result from [4] concerns IS-executions of a k-set agreement algorithm in wait-free shared memory with $k < n$ and $x_i = i$ initially.

The the proof consists of an induction on the correct processes, denoted the *participating set* of processes. The induction starts from participating set p_1 and involves the induction step from participating set $\{p_1, \dots, p_\ell\}$ to $\{p_1, \dots, p_\ell, p_{\ell+1}\}$. The maximal participating set is obviously $\{p_1, \dots, p_n\} = \Pi$. It is shown that *there is an odd number (greater than or equal to* 1*) of executions where there are* ℓ *distinct decision values, with participating set* $\{p_1, \dots, p_\ell\}$*, for all values of* ℓ. Because we may set $\ell = n$ and $n > k$, this yields the existence of an execution where more than k values are decided upon.

While the base case of the induction is rather obvious – a single process will decide on a single value in the wait-free setting – the induction step contributes the bulk of the proof. In short, for the induction step from ℓ to $\ell + 1$, a graph is constructed, where each vertex corresponds to an execution of the set $S_{\ell+1}$, i.e. those executions with participating set $\{p_1, \dots, p_{\ell+1}\}$. Additionally, there is one so-called imaginary vertex v^* in the graph that does not correspond to an execution. The presence of an edge between two vertices (that both represent an execution from the set $S_{\ell+1}$) is determined by the relation in which the two respective executions stand[3]:

- An edge is present between two executions, if and only if exactly one process p observes a difference between the two and in both executions ℓ distinct decisions from the set $\{1, \dots, \ell\}$ are made by those processes for which the executions are indistinguishable. The edge is said to be with respect to p.
- An edge is present between an execution and v^*, if and only if $p_{\ell+1}$ is unseen in the execution and ℓ distinct decisions from the set $\{1, \dots, \ell\}$ are made by the processes of $\{p_1, \dots, p_\ell\}$ in the execution. The edge is said to be with respect to $p_{\ell+1}$.

Using the induction hypothesis, it is shown that the imaginary vertex has odd degree: Recall that the induction hypothesis states that there is an odd number of executions in S_ℓ, where ℓ distinct decisions are made. Picking any one of them, it is clear that we can simply append the suffix $\{p_{\ell+1}\}^r$ where only $p_{\ell+1}$ takes steps until $p_{\ell+1}$ decides in order to create an execution adjacent to v^*. By the determinism of $p_{\ell+1}$, r is unique and thus the number of executions adjacent to v^* is odd as a result of this simple one-to-one and onto mapping.

The proof is concluded by showing that the executions of $S_{\ell+1}$ where $\ell + 1$ distinct values are decided (i.e. values from $\{1, \dots, \ell + 1\}$, as we had $x_i = i$ initially) have degree 1 while all other executions have degree 0 or 2.

- The former can be intuitively understood as follows: Let q be the process that decides $\ell + 1$. Because the participating set contains $\ell + 1$ processes, each processes different from q must decide on a distinct value from $\{1, \dots, \ell\}$. It follows that q is the only process with respect to which there can be an edge.

[3]In the following presentation, we do not distinguish between executions and edges that represent these executions, i.e., we say for example that an execution and an edge are incident when the graph contains an edge that includes the vertex representing this execution.

If q is seen, by Lemma 4, there is precisely one execution indistinguishable for all processes but q, thus ensuring that both executions are the only ones adjacent to each other in the graph.

If q is unseen, then by Lemma 5 there is no execution indistinguishable for all processes except q. Therefore, since the processes other than q decide $1, \dots, \ell$ and $x_i = i$ initially, by validity it must hold that $q = p_{\ell+1}$. Hence, there is precisely one edge from this execution, namely the one to v^*.

- The latter can be shown by first noting that executions where the the set of decision values is different from $\{1, \dots \ell\}$ cannot have an incident edge, since obviously no subset of the processes of such executions decides all the values from $\{1, \dots, \ell\}$.

  It remains to be shown what happens when the set of decisions in an execution is $\{1, \dots \ell\}$. As the number of processes participating in this execution is $\ell + 1$, two processes q_1 and q_2 must decide on the same value u. Note that all processes different from q_1, q_2 decide on a distinct decision value from $\{1, \dots, \ell\} \setminus \{u\}$. Just as before, it follows immediately that q_1, q_2 are the only processes with respect to which there can be an edge. Again, just as before, we identify two distinct cases:

  If q_1 and q_2 are seen, then by Lemma 4 there is precisely one edge with respect to q_1 and one edge with respect to q_2 and no other edges from this execution.

  Note that two unseen processes constitute a contradiction. Thus, the remaining case is that either q_1 or q_2, say q_2 is unseen. In this scenario, by Lemma 4, there is precisely one execution indistinguishable for all process but q_1 and thus there is an edge with respect to q_1 to this execution.
  Moreover, as q_2 is unseen and each of the ℓ processes other than q_2 decides on a distinct value from $\{1, \dots, \ell\}$, because we had $x_i = i$, it follows from validity that $q_2 = p_{\ell+1}$. Because of this and because of Lemma 5, there is a single edge with respect to q_2 to v^*. Together with the single edge with respect to q_1 we have that the current execution has degree two.

Since, by the handshake lemma, any graph has an even number of vertexes with odd degree and v^* has odd degree and additionally the only other vertexes with odd degree are the ones where $\ell + 1$ values are decided, there must exist an odd number, greater than or equal to one, of executions with $\ell + 1$ decisions.

7.4 Comparing the BRS-Theorem and Counting Arguments

Let us now compare the proof of the impossibility of wait-free k-set agreement from [4], as sketched in the previous section, and a similar non-topological proof for $(n-1)$-set Agreement from [5] with the BRS-Theorem. What is immediately apparent is that the BRS-Theorem, while highly generic, requires for one of its core arguments that the distributed system can be partitioned in such a way that there exist processes that have no knowledge about some other processes and vice-versa. It is this partitioning that provides us with the fact that these processes from a different partition cannot decide on the same value in runs where the initial values of the partitions form disjoint sets. This, in conjunction with a clever application of the impossibility of consensus in a restricted system, is the crux of the BRS-Theorem. As we have seen throughout this thesis, this argument can be applied in a great variety of scenarios. Nevertheless, we have not yet found a way to use the same argument in the shared memory model, as elaborated previously in this chapter.

The counting-based impossibility proofs are in a way a non-topological version of the original topological impossibility proofs of k-set agreement from [14, 19, 32]: they show elegantly how to express the application of Sperner's Lemma, which is the topological theorem to which the original topological impossibility results all share a connection, using purely combinatoric arguments [3]. To the best of our knowledge, however, as of the writing of this thesis, there are no counting-based proofs that show the tight failure bound for the impossibility of k-set agreement, i.e., the k-resilient impossibility with up to k failures. Instead, they require the much more restrictive assumption of wait-freedom, i.e., they show the impossibility of $(n-1)$-resilient k-set agreement. Wait-freedom, however, in turn easily allows an arbitrary partitioning to occur, which is again in the realm of the BRS-Theorem (Theorem 1). We hence conjecture that these are indeed (at least) two different and independent sources of impossibility for general k-set agreement $(1 < k < n)$, namely, partitioning (covered by the BRS-Theorem) and higher-order "symmetry breaking" (covered by the topological proofs).

Finally, we would like to note that as far as we know, it is not yet known whether the partitioning (hence the BRS-Theorem) has a topological counterpart, like the counting-based proofs presented here obviously have. Finding such a topological analogy for the BRS-Theorem could indeed establish a relation to the counting-based approaches that is not yet apparent.

CHAPTER 8

Summary and Outlook

In this thesis, we investigated and extended the range of applications of the recently published BRS-Theorem (Theorem 1). The BRS-Theorem combines classic partitioning arguments with the impossibility of consensus, using a rigorous framework of systems and their subsystems, restrictions of algorithms and compatibility of runs.

In several chapters, we have applied the BRS-Theorem to a wide variety of different system models:

1. We used it to derive new non-topological impossibilities for k-set agreement in synchronous dynamic message-passing networks, where we showed that root components must be stable for sufficiently long, and that even an eventual stability of all root components may still lead to a safety violation. In [45], this impossibility has been employed to develop assumptions and an algorithm that allows to solve k-set agreement in this setting.

2. We investigated a more fundamental description of synchronous dynamic networks, namely networks where some low-level graph predicate $\mathcal{P}_{\text{srcs}}$ holds. We introduced a rigorous proof based on the BRS-Theorem in conjunction with existing results on consensus with lossy links, and showed that the slightly weaker predicate $\mathcal{P}^{r}_{\text{srcs}}$ makes solving k-set agreement impossible.

3. We proceeded to message-passing systems prone to general omission failures, a failure model that includes crash failures as a special case, but is in general more severe. We employed the BRS-Theorem and some well-known consensus impossibility results to give a (non-topological) lower bound that matches the lower bound for crash failures.

4. We elaborated on the challenges of adopting the BRS-Theorem to the more powerful shared memory model of computation, where topological proofs are usually used for proving k-set impossibilities. We observed a fundamental problem related to the fact that the partitioning required for applying the BRS-Theorem cannot usually occur there. We hence conjecture that there are actually two different sources of k-set impossibilities.

We hope to find answers to the following still open questions in future research:

- Can we use the BRS-Theorem in the search for the weakest failure detector for k-set agreement in message passing systems?

  We have seen how the BRS-Theorem can be employed to show that the failure detector (Σ_k, Ω_k) is too weak to solve k-set agreement in message passing systems. On the other hand, the slightly stronger failure detector (Σ_k, X_k) is sufficient, but unfortunately too strong. This implicates that the weakest failure detector is somewhere between (Σ_k, Ω_k) and (Σ_k, X_k). As we were able to show the insufficiency of (Σ_k, Ω_k) using the BRS-Theorem, analyzing the detailed reasons for it to apply, might be the key that yields the necessary property to be added to (Σ_k, Ω_k) to make it the weakest failure detector for k-set agreement in message passing systems.

- What is the weakest assumption that makes k-set agreement possible in the DDN model when eventual stability is the only assumption (apart from assumptions that exclude trivial impossibilities)?

  We developed a sufficient and very weak assumption and an according algorithm, which are, at the time of writing, under review for publication. Yet, it remains open whether the impossibility from Theorem 7 can be strengthened to precisely match this assumption or whether this assumption can be weakened any further.

- What is the weakest predicate that enables k-set agreement in the DDN model?

  We saw that $\mathcal{P}_{\text{srcs}}(k)$ is sufficient, yet were not able to show that it is the weakest predicate that provides solvability. Rather, it seems that the implications of $\mathcal{P}_{\text{srcs}}(k)$ are relatively strong, which suggests that a weaker predicate exists. On the other hand, $\mathcal{P}^r_{\text{srcs}}(k)$, which is only slightly weaker than $\mathcal{P}_{\text{srcs}}(k)$, was shown to be insufficient for making k-set agreement solvable. This indicates that there might exist a third predicate, stronger than $\mathcal{P}^r_{\text{srcs}}(k)$ but weaker than $\mathcal{P}_{\text{srcs}}(k)$, that precisely captures the solvability boundary.

- Is $t - k + 2$ a tight lower bound for the round complexity of k-set agreement algorithms in systems with t general omission failures?

  We have shown how an application of the BRS-Theorem leads to the impossibility of k-set agreement with t omission failures in less than $\frac{t}{k}$ rounds. Moreover, we mentioned that for uniform algorithms, the BRS-Theorem supplies us with the impossibility in $t - k + 2$ rounds. However, the issue whether $t - k + 1$ rounds constitute an impossibility even for non-uniform algorithms, requires further investigation.

- Is there a topological counterpart of the BRS-Theorem?

  We have seen that the BRS-Theorem is not able to provide impossibilities in systems where no suitable partitioning can be established. One approach to circumvent this shortcoming would be to find a transformation of the BRS-Theorem to a topological counterpart and analyze on a topological level what is needed to establish said impossibilities. This might lead to a widely applicable and novel argument for the impossibility of k-set agreement in general.

Bibliography

[1] Marcos K. Aguilera. Stumbling over Consensus Research: Misunderstandings and Issues. In Bernadette Charron-Bost, Fernando Pedone, and André Schiper, editors, *Replication*, volume 5959 of *Lecture Notes in Computer Science*, pages 59–72. Springer Berlin Heidelberg, 2010.

[2] Marcos K. Aguilera and Sam Toueg. A simple bivalency proof that -resilient consensus requires rounds. *Information Processing Letters*, 71(3-4):155–158, August 1999.

[3] Hagit Attiya. A Direct Lower Bound for k-Set Consensus. In *PODC*, 1998.

[4] Hagit Attiya and Armando Castañeda. A non-topological proof for the impossibility of k-set agreement. *Theoretical Computer Science*, September 2012.

[5] Hagit Attiya and Ami Paz. Counting-Based Impossibility Proofs for Renaming and Set Agreement. In Marcos K. Aguilera, editor, *Distributed Computing*, volume 7611 of *Lecture Notes in Computer Science*, pages 356–370. Springer Berlin Heidelberg, 2012.

[6] Hagit Attiya and Sergio Rajsbaum. The Combinatorial Structure of Wait-Free Solvable Tasks. *SIAM Journal on Computing*, 31(4):1286–1313, January 2002.

[7] Hagit Attiya and Jennifer Welch. *Distributed Computing (Second Edition) Fundamentals, Simulations and Advanced Topics*. Wiley-Interscience, 2004.

[8] Martin Biely, Peter Robinson, and Ulrich Schmid. Easy Impossibility Proofs for k-Set Agreement in Message Passing Systems. In Antonio Fernàndez Anta, Giuseppe Lipari, and Matthieu Roy, editors, *Principles of Distributed Systems*, volume 7109 of *Lecture Notes in Computer Science*, pages 299–312. Springer Berlin Heidelberg, 2011.

[9] Martin Biely, Peter Robinson, and Ulrich Schmid. Solving k-Set Agreement with Stable Skeleton Graphs. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 1488–1495. IEEE, May 2011.

[10] Martin Biely, Peter Robinson, and Ulrich Schmid. Agreement in Directed Dynamic Networks. *arXiv:1204.0641*, 7355:73–84, April 2012.

[11] Martin Biely, Peter Robinson, and Ulrich Schmid. Weak synchrony models and failure detectors for message passing k-set agreement. *IEEE Transactions on Parallel and Distributed Systems*, 2013. (to appear).

[12] Martin Biely, Peter Robinson, Ulrich Schmid, and Kyrill Winkler. Easy Impossibility Proofs for k-Set Agreement. (in preparation).

[13] François Bonnet and Michel Raynal. On the road to the weakest failure detector for k-set agreement in message-passing systems. *Theoretical Computer Science*, 412(33):4273–4284, July 2011.

[14] 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*, STOC '93, pages 91–100, New York, NY, USA, 1993. ACM.

[15] Tushar D. Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[16] Tushar D. Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996.

[17] Bernadette Charron-Bost and André Schiper. Uniform consensus is harder than consensus. *Journal of Algorithms*, 51(1):15–37, April 2004.

[18] S. Chaudhuri, M. Herlihy, Nancy A. Lynch, and Mark R. Tuttle. A tight lower bound for k-set agreement. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 206–215. IEEE, November 1993.

[19] Soma Chaudhuri. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems. *Information and Computation*, 105(1):132–158, July 1993.

[20] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. Tight failure detection bounds on atomic object implementations. *J. ACM*, 57:22:1–22:32, May 2010.

[21] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the 23rd ACM Symposium on Principles of Distributed Computing (PODC'04)*, pages 338–346. ACM Press, 2004.

[22] Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, January 1987.

[23] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, October 1990.

[24] Danny Dolev and H. Raymond Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, 12(4):656–666, November 1983.

[25] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, April 1988.

[26] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, September 2003.

[27] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[28] Carole D. Gallet, Hugues Fauconnier, and Rachid Guerraoui. Tight failure detection bounds on atomic object implementations. *J. ACM*, 57(4), May 2010.

[29] Carole D. Gallet, Hugues Fauconnier, Rachid Guerraoui, Vassos Hadzilacos, Petr Kouznetsov, and Sam Toueg. The weakest failure detectors to solve certain fundamental problems in distributed computing. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 338–346, New York, NY, USA, 2004. ACM.

[30] Jim N. Gray. Notes on data base operating systems. In R. Bayer, R. M. Graham, and G. Seegmüller, editors, *Operating Systems*, volume 60 of *Lecture Notes in Computer Science*, pages 393–481. Springer Berlin Heidelberg, 1978.

[31] Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In Sape Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 2nd edition, 1993.

[32] 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*, STOC '93, pages 111–120, New York, NY, USA, 1993. ACM.

[33] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[34] Nancy A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, 1996.

[35] Achour Mostefaoui, Sergio Rajsbaum, Michel Raynal, and Corentin Travers. On the computability power and the robustness of set agreement-oriented failure detector classes. *Distributed Computing*, 21(3):201–222, September 2008.

[36] Achour Mostéfaoui, Michel Raynal, and Julien Stainer. Relations linking failure detectors associated with k-set agreement in message-passing systems. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *SSS*, volume 6976 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2011.

[37] Gil Neiger. Failure detectors and the wait-free hierarchy (extended abstract). In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*, PODC '95, pages 100–109, New York, NY, USA, 1995. ACM.

[38] Philippe R. Parvédy and Michel Raynal. Optimal early stopping uniform consensus in synchronous systems with process omission failures. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '04, pages 302–310, New York, NY, USA, 2004. ACM.

[39] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *Software Engineering, IEEE Transactions on*, SE-12(3):477–482, March 1986.

[40] Michel Raynal. *Communication and agreement abstractions for fault-tolerant asynchronous distributed systems*. Morgan & Claypool Publishers, 2010.

[41] Michel Raynal and Corentin Travers. Synchronous Set Agreement: a Concise Guided Tour (including a new algorithm and a list of open problems). In *PRDC '06. 12th Pacific Rim International Symposium on Dependable Computing*, pages 267–274. IEEE, December 2006.

[42] Michael 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*, STOC '93, pages 101–110, New York, NY, USA, 1993. ACM.

[43] Nicola Santoro and Peter Widmayer. Time is not a healer. In *Proc. 6th Annual Symposium on Theor. Aspects of Computer Science (STACS'89)*, LNCS 349, pages 304–313, Paderborn, Germany, February 1989. Springer-Verlag.

[44] Ulrich Schmid, Bettina Weiss, and Idit Keidar. Impossibility Results and Lower Bounds for Consensus under Link Failures. *SIAM Journal on Computing*, 38(5):1912–1951, January 2009.

[45] Manfred Schwarz. k-Set Agreement in Graphs With Stable Root Components. Master's thesis, Technische Universität Wien, Karlsplatz 13, 1040 Wien, 2013.

[46] Manfred Schwarz, Kyrill Winkler, Martin Biely, Peter Robinson, and Ulrich Schmid. k-set agreement under dynamic link failures. (submitted).