The Unified Structure of Consensus: a *Layered Analysis* Approach

Yoram Moses*
Department of Applied Math and CS,
The Weizmann Institute of Science,
Rehovot, 76100 Israel.
Email: yoram@cs.weizmann.ac.il

Sergio Rajsbaum†
Instituto de Matemáticas, UNAM,
Ciudad Universitaria, D.F. 04510, México.
Email: rajsbaum@servidor.unam.mx

Abstract

We introduce a simple notion of *layering* that provides a tool for defining submodels of a given model of distributed computation. We describe two layerings, the *synchronic* and the *permutation* layering, and show that they induce appropriate submodels of several asynchronous models of computation. The synchronic layering applies to the synchronous model too.

We perform a model-independent analysis of the consensus problem in terms of abstract connectivity properties of layering functions. By defining particular layerings in specific models, we derive several popular (and some new) lower bounds and impossibility results for consensus in various classical models. These results are often stronger in the sense that they apply to the submodel induced by the layering. The proofs obtained in this way are also simpler and more direct than existing ones. Moreover, the analysis is done in a uniform fashion and demonstrates the fundamental common structure of the consensus problem in the presence of failures.

The analysis is then extended to general decision problems (1-resilient in the asynchronous models, t-rounds in the t-resilient synchronous model), providing a characterization of solvability of decision problems in the style of [8] which, for some of the models, is given for the first time.

1 Introduction

For almost two decades now, the consensus problem has played a central role in the study of fault-tolerant distributed computing, e.g. [23, 13, 12, 10, 14, 20, 16, 8, 9]. It has clearly received the greatest amount of attention in the theoretical literature on distributed computing, and has been studied in a large variety of models and under many types of failure assumptions. Work on different variants often involved related notions, especially when proving impossibility results; mainly a notion of some form of connectivity defined in terms of a relation on pairs of states that are indistinguishable to some of the processes. However, proofs for different models are often based on distinct and somewhat ad hoc techniques. In particular, there have been considerable differences between the study of consensus in asynchronous models, and its study in synchronous and partially synchronous ones.

In order to cope with the proliferation of distributed computing models, researchers have proposed a variety of simulations between models. The aim is to establish a relation (often of equivalence) between the possibility of solving problems of certain types in different models. This is used to establish impossibility results in particular models, or to provide a systematic way to transform protocols written in one model into protocols for another model. Various such simulations have been given, e.g. between shared memory and message passing [3]; between snapshot shared memory and read/write shared memory [2], or between immediate snapshot shared memory and read/write shared memory [5, 6]; between synchronous and asynchronous message passing [1]; between two shared memory models of different resilience [5].

This paper attempts to present a uniform approach to study solvability of consensus and other decision problems in various models of computation, where crash failure behavior is possible (e.g., sending omissions or Byzantine failures: A faulty processor can fail to send messages altogether from some point on, and thus behave as if it has crashed). We start by considering an abstract model based on the runs of a distributed system, and a failure model. Then we introduce the notion of *layering* which allows to focus only on interesting states among a subset of the runs. We think of a "layer" as a set of (not necessarily immediate) successor states of a given state.

Roughly, we use layering in the following sense. Given a model of distributed computation, we identify particular legal sequences of actions for the scheduler, which we think of as generating a "layer". We require that performing such layers repeatedly starting from a legal initial state will result in a legal run in the model. Thus, in a precise sense, such a layering can be viewed as defining a submodel of the original model.[1] Any protocol for the original model translates directly to one in the submodel. Moreover, the model

*This work has been supported by a Helen and Milton A. Kimmelman career development chair.

†Part of this work was done while visiting the Weizmann Institute of Science. Partially supported by Conacyt and DGAPA-UNAM Projects.

[1]Layering saves us the trouble of explicitly defining the submodel as a model of computation with a new transition function, new actions, etc.

and submodel generally share many features. In particular, lower bounds and impossibility results proven for the submodel typically translate directly into the original model. The use of layerings facilitates performing round-by-round analysis: Almost all of our results regarding consensus will follow from analyzing a single layer of computation. The benefit of working in a submodel with a simpler structure than that of the original model is well known; some recent examples are [4, 5, 6, 8, 19, 25].

In the main part of the paper we concentrate on the consensus problem. First, we perform an abstract and model-independent analysis of consensus using layering. Based on this analysis, implications for specific models are obtained by demonstrating that appropriate layerings can be defined in the model. We exemplify the approach on the following 1-resilient models: asynchronous shared memory, asynchronous message passing, mobile failures, and in the t-resilient synchronous message passing model.

We describe two specific layerings: the *synchronic* and the *permutation* layering, and show that they apply to the first three models. The synchronic layering defines a submodel of the asynchronous models that is very close in structure to being synchronous. Indeed, we show that the synchronic layering applies to the synchronous message passing model too. The permutation layering is inspired by the immediate snapshot wait-free model of [5], although we define it both for the message passing and for the shared memory models, 1-resilient. This appears to be the first variant of the immediate snapshot model suggested for a message-passing model.

Regarding consensus, we provide:

- New, simple and uniform lower bound and impossibility proofs for the standard synchronous and asynchronous models. In particular, we show a bivalence-based proof for the synchronous case, and a direct round-by-round construction of a bivalent run (not employing a critical state argument) in asynchronous cases.

- Stronger impossibility results with respect to submodels of the full asynchronous models, in which there is only a small degree of asynchrony. These demonstrate how little asynchrony is needed to make consensus unsolvable.

- Finally, we show that the asynchronous models are equivalent in terms of the 1-resilient solvability of decision tasks. In particular, the slightly asynchronous submodels are no more powerful than the fully asynchronous ones. Moreover, in the full paper we show that in a sense, they are strictly stronger than what can be done t-resiliently in t rounds of the standard synchronous model. Thus, our results can be viewed as providing characterizations of solvability of decision problems in the style of [8] which, for the mobile failure and synchronous models, are new.

We consider the layering technique to be useful in a number of ways:

- It provides a tool for performing model-independent round-by-round analysis of decision tasks and related problems.

- Results are obtained directly, and not by means of specially tailored reductions.

- Popular topological treatments (e.g. [18, 25, 5, 4]) focus on the local *final* states of processes. We consider states at intermediate stages of the computation as well. Moreover, the global state also takes into account the state of the environment, and can therefore treat both shared memory and message-passing models.

- We make use of a novel notion of *valence connectivity* of a set of states, whose definition depends on the decisions taken by the protocol in the possible futures of these states. In addition we use the more traditional notion of connectivity based on indistinguishability. The combined used of the two notions proved very useful for unifying the analysis of consensus in the synchronous and asynchronous models.

Our analysis in this paper concentrates on the consensus problem and on 1-resilient solvability. By focusing on these "basic" cases we obtain a direct and uniform analysis of these questions in simple and elementary terms. The proofs of all of our results are short, which suggests that the notions we use are fairly natural. We believe that, with small modifications, the same type of analysis and style of reasoning can be applied to study more general problems involving higher levels of topological connectivity.

Two other papers that attempt to unify synchronous and asynchronous models in a round by round style are: [15] via failure detectors in shared memory, and [17] using topological techniques in synchronous, asynchronous and partially synchronous message passing systems, with applications to set-consensus. A different abstract model based on the runs of a distributed system is proposed in [21].

For lack of space, some of the proofs are omitted or just sketched in this extended abstract.

2 Definitions

In this paper we wish to present an approach that is relevant in a number of different models. We therefore start out by defining those aspects of the models that the technique will use, then follow it by presenting the basic approach in these more general terms, and differ some issues to be discussed only when we come to applications in concrete models.

Systems

Throughout the paper we will assume there is a fixed finite set of $n \geq 2$ processes, which we shall denote by $1, 2, \ldots, n$, and an *environment*, denoted by e, which is responsible for everything else that happens.

The (global) state of a system consists of a local state for each one of the processes, as well as a local state for the environment, which captures all else that is relevant to the operation of the system. Thus, aspects such as messages in transit, or the values of shared variables, will typically be represented in the environment's state. More formally, for every $i \in \{e, 1, 2, \ldots, n\}$, we assume there is a set L_i consisting of all possible *local states* for i. The set of *global states*, which we will simply call *states*, will consist of

$$\mathcal{G} = L_e \times L_1 \times \cdots \times L_n.$$

If x is a state, then we denote by x_i the local state of i in the state x. We say that two states x and y *agree modulo* j if $x_e = y_e$ and $x_i = y_i$ for all processes $i \neq j$.

A *run* over $\mathcal{G}$ is a function r from the non-negative integers to $\mathcal{G}$ defining an infinite sequence of states of $\mathcal{G}$. The state $r(0)$ is called the *initial state* of r. An *execution* is a finite or infinite subinterval of a run.

A *system* $\mathcal{R}$ is a set of runs. With respect to such a system, a state y is said to *extend* the state x if there is a finite execution that starts in x and ends in y. By definition, x extends x for every state x of the system. An *infinite execution extending* x is the suffix starting at x of a run in which x appears.

Finally, we will usually be interested in systems that are in a precise sense derived from some automaton whose states are the states that appear in the runs of our system. Intuitively, this means that all aspects of the run up to a given state that may affect the future of the run are somehow captured in the state. Formally, we will capture this by assuming such a property directly. We say that a system $\mathcal{R}$ is *admissible* if it satisfies the following *pasting* condition: For every state x, if there are runs $r, r' \in \mathcal{R}$ and indices $m, m' \geq 0$ such that $r(m) = r'(m') = x$, then there is a run $\hat{r} \in \mathcal{R}$ such that

$$\hat{r}(k) = \begin{cases} r(k) & \text{for } k \leq m, \text{ and} \\ r'(m' + (k - m)) & \text{for } k > m. \end{cases}$$

All systems we consider in this paper will be assumed to be admissible.

Failures

A large variety of types of faulty behavior have been proposed and studied in the literature. In order to avoid the need to commit to particular types of failures, we identify two abstract properties about models allowing crash failures, which are all that we need for our analysis. One assumption will involve a notion of independence of failures, while the other will capture an observable aspect of crash failures.

For every system $\mathcal{R}$ appearing in this paper, we assume that there is an associated function Faulty : $\{1, \ldots, n\} \times \mathcal{R} \rightarrow \{\textbf{Yes}, \textbf{No}\}$ that, given a process name i and a run r, will say whether or not i is faulty in r.[2] A function Faulty defined for a system $\mathcal{R}$ induces a notion of who is *failed at a state* (with respect to $\mathcal{R}$). We say that a process i is *failed at state* x if i is faulty in all runs of $\mathcal{R}$ in which x appears. Otherwise i is *non-failed* at x. The one property we require of such a function Faulty is

Fault Independence: *For every state x of $\mathcal{R}$ there is a run $r^x \in \mathcal{R}$ in which x appears, such that the only processes that fail in r^x are those that are already failed at x.* Formally, this is captured by $\forall i.$ [Faulty(i, r^x) iff i is failed at x]. When modelled in an appropriate way, all failure models we are aware of satisfy this condition.

Intuitively, once a process has crashed, it's state and behavior can no longer affect the state of the rest of the system. Motivated by this, the following definition captures an abstract property of crash failures. We say that a system $\mathcal{R}$ *displays an arbitrary crash failure* with respect to a set X of states of $\mathcal{R}$ if the following condition holds. For every $x, y \in X$ and process j if x and y agree modulo j, then there exist in $\mathcal{R}$ runs r^x and r^y and times $m_x, m_y \geq 0$ such that (i) $r^x(m_x) = x$ and $r^y(m_y) = y$, (ii) $r^x(m_x + k)$ and $r^y(m_y + k)$ agree modulo j for all $k \geq 0$, and (iii) Every process $i \neq j$ that is not failed in x and in y is nonfaulty in r^x and in r^y.

[2]The function Faulty determines who is faulty in a run as a function of the whole infinite run. Obviously, in some models it is possible to determine that a process is faulty by considering only a prefix of the run, sometimes even a single state. In other cases, however, it is impossible to determine this after a finite amount of time. (See, e.g., the asynchronous systems in [14].)

3 Consensus and Connectivity

In the classical binary consensus problem [23], processes start with binary values and have to decide on binary values satisfying three conditions. *Decision,* requires that every nonfaulty process eventually decides; *Agreement* requires that the decisions are identical; and *Validity* requires that each decision was somebody's input. A decision is assumed written into a write-once variable d_i ever-present, and initially undefined, in i's local state. The 2^n possible initial states for consensus are defined formally, and the set of these states is denoted by Con_0. We defer this formalization to the full version.

In the sequel, we shall focus on systems that are compatible with the consensus problem. We call a system $\mathcal{R}$ a *system for consensus* if (i) the set of initial states in $\mathcal{R}$ consists exactly of Con_0, (ii) the local state x_i of every process i in a state x of $\mathcal{R}$ contains the variable d_i, and the runs all satisfy that d_i is write-once, and (iii) in the associated Faulty function for $\mathcal{R}$ no process is failed at an initial state of a run of $\mathcal{R}$. In Sections 3–6, all systems referred to will be assumed to be systems for consensus.

Decisions and valence

It has long been recognized [14] that considering what the possible decisions in the future of a given state are can be a useful tool for analyzing the structure of the consensus problem. This gives rise to the following notions of the *valence* of a state. With respect to a given system $\mathcal{R}$, we define a state x in $\mathcal{R}$ to be *v-valent* if there is an execution of $\mathcal{R}$ extending x in which at least one nonfaulty process decides v. The state x will be called *v-univalent* if it is v-valent and, for all $v' \neq v$, it is not v'-valent. Since we are considering only binary decisions, we say that x is *bivalent* if it is both 0-valent and 1-valent. One useful consequence of the agreement property is captured in the following lemma. We say that a process i *has decided by x* if in the state x we have that $d_i \neq \perp$.

Lemma 3.1 *Let $\mathcal{R}$ be a system satisfying the agreement requirement and assume that no more than $t < n$ processes fail in runs of $\mathcal{R}$. If x is a bivalent state of $\mathcal{R}$ then at least $n - t$ non-failed processes at x have not decided by x.*

Proof: Since x is bivalent, there is a run r^0 extending x in which at least one nonfaulty process decides 0. The set P_0 of nonfaulty processes in r^0 consists of at least $n - t$ processes, they are all non-failed at x, and by the agreement property none of them has decided 1 by x. Using symmetric reasoning, we obtain the existence of a set P_1 of at least $n - t$ non-failed processes at x that have not decided 0 by x. By the failure independence property, there is a run $\hat{r}$ extending x in which the only processes that fail are the ones that are already failed at x. All processes in $P_0 \cup P_1$ are nonfaulty in $\hat{r}$. By the agreement property, there is at most one value $v \in \{0, 1\}$ on which nonfaulty processes decide in $\hat{r}$. If nonfaulty processes do not decide 0 in $\hat{r}$, then no process in P_0 has decided by x, and if they do not decide 1 in $\hat{r}$, then no process in P_1 has decided by x. In either case, the claim holds. ■

Lemma 3.1 provides us with a tool for showing when a consensus protocol cannot terminate its operation altogether. This is useful for the study of lower bounds for the synchronous model which we treat in Section 6. In asynchronous systems it is usually the case that even when a process is faulty, this faultiness cannot be determined in finite time. We say that a system $\mathcal{R}$ displays *no finite failure* if, for all states x of $\mathcal{R}$, no process is failed at x. For such systems, a proof similar to that of Lemma 3.1 in fact shows:

Lemma 3.2 *Let $\mathcal{R}$ be a system that displays no finite failure and satisfies the agreement requirement. If the state x of $\mathcal{R}$ is bivalent, then no process has decided by x.*

Connectivity

As has been observed by many authors starting with [13], connectivity plays a central role in the analysis of decision problems. We use two types of connectivity:

Definition 3.1 *Let x and y be states of $\mathcal{R}$. With respect to $\mathcal{R}$ we define*

Similarity: *x and y are* similar*, denoted by $x \sim_s y$, if there exists a process j such that (i) the states x and y agree modulo j, and (ii) there exists a process $i \neq j$ that is non-failed in both x and y.*

Shared valence: *x and y have a* shared valence*, denoted by $x \sim_v y$, if for some $w \in \{0,1\}$ both x and y are w-valent.*

A very useful sufficient condition that guarantees that states have a shared valence is given by:

Lemma 3.3 *Let $\mathcal{R}$ satisfy the decision requirement, let X be a set of states of $\mathcal{R}$, and assume that $\mathcal{R}$ displays an arbitrary crash failure with respect to X. For all $x, y \in X$, if $x \sim_s y$ then $x \sim_v y$.*

The argument for this lemma is well known. Intuitively, if x and y agree modulo j, then by crashing j at both x and y we obtain runs r^x and r^y that are indistinguishable to the nonfaulty processes, who in turn end up deciding on the same value v in both runs.

For every set X of states, each of $\sim_s$ and $\sim_v$ is in particular a binary relation on X. We can thus view $(X, \sim_s)$ and $(X, \sim_v)$ as the corresponding graphs. The set X is said to be *valence connected* if the graph $(X, \sim_v)$ is connected, and X is said to be *similarity connected* if $(X, \sim_s)$ is connected. Similarity connectedness has often been considered in the literature (see, e.g., [24]). Valence connectedness is, to the best of our knowledge, new. In general, valence connectedness is not a very strong condition. Indeed, X is valence connected exactly if either (i) for some value v, all states of X are v-univalent, or (ii) there exists at least one bivalent state in X. Thus, valence connectedness can be used to demonstrate the existence of bivalent states:

Lemma 3.4 *Assume $\mathcal{R}$ satisfies the decision requirement and X is valence connected. If X contains both 0-valent and 1-valent states, then there is a bivalent state in X.*

Lemma 3.3 directly implies the following.

Lemma 3.5 *Let $\mathcal{R}$ satisfy the decision requirement and let X be similarity connected. If $\mathcal{R}$ displays an arbitrary crash failure with respect to X, then X is valence connected.*

Lemma 3.4 and Lemma 3.5 allow us to reprove in our setting the well-known fact from [14] that when even a single process can crash, there must be a bivalent initial state for consensus.

Lemma 3.6 *The set Con_0 is similarity connected. Moreover, if $\mathcal{R}$ satisfies the decision requirement and displays an arbitrary crash failure with respect to Con_0, then Con_0 is also valence connected. Finally, if in addition the validity requirement holds, then there is a bivalent initial state in Con_0.*

Proof: To prove that Con_0 is similarity connected we will show that every two initial states $x, y \in \mathsf{Con}_0$ are connected by a sequence of states each pair of which are similarity connected. Choose $x, y \in \mathsf{Con}_0$. For $0 \leq l \leq n$, define x^l by setting

$$x_j^l = \begin{cases} x_j & \text{for } j = e \text{ and all } j \leq l; \text{ and} \\ y_j & \text{for all } j > l. \end{cases}$$

Clearly, $x^l \in \mathsf{Con}_0$, and it is easy to check that $x^0 = x$ and $x^n = y$. (Recall that $x_e = y_e$ by definition of Con_0.) Moreover, for every $0 < l \leq n$ we have that in x^{l-1} and x^l agree modulo l, since the local states of the environment and of all processes, except possibly that of l, are equal. Since $n \geq 2$, the states x and y are similarity connected. Since $\mathcal{R}$ satisfies the decision requirement and displays an arbitrary crash failure with respect to Con_0, Lemma 3.5 implies that Con_0 is valence connected. Now, if in addition we have the validity requirement, then the state $x^0 \in \mathsf{Con}_0$ in which all processes start with initial value 0 is 0-valent, while the symmetric state x^1 in which they all start with value 1 is 1-valent. It now follows by Lemma 3.4 that there is a bivalent state in Con_0. ■

4 Layering

In some cases, the set of runs of a given protocol in a particular model can be very rich. We often want to consider properties of a model that are already evident in a subset of the runs of the protocol, in which the environment's behavior may have a nicer structure. For example, we will later on consider runs of an asynchronous system that can be divided into what are approximately synchronous rounds.

Let $\mathcal{G}$ be a set of states. A function $S : \mathcal{G} \to 2^{\mathcal{G}} \setminus \{\emptyset\}$ is called a *successor function* for $\mathcal{G}$. A run r over $\mathcal{G}$ is called an *S-run* if $r(m+1) \in S(r(m))$ for all $m \geq 0$. When focusing on runs compatible with the consensus problem, we have $\mathsf{Con}_0 \subseteq \mathcal{G}$ and we can view a successor function S as generating a system $\mathcal{R}_S$ consisting of the set of all S-runs with initial state in Con_0. The essential property that will later on yield lower bound and impossibility results in a variety of models is captured in the following lemma.

Lemma 4.1 *Let S be a successor function, and assume $\mathcal{R}_S$ satisfies decision. If x is bivalent in $\mathcal{R}_S$ and $S(x)$ is valence connected, then there exists a bivalent state $y \in S(x)$.*

Proof: Since x is bivalent, at least one of the states in $S(x)$ is 0-valent and at least one is 1-valent. By assumption, the set $S(x)$ is valence connected. Thus, by Lemma 3.4 we have that there is a bivalent state $y \in S(x)$. ■

Given Lemma 4.1, if $S(x)$ is valence connected for every $x \in \mathcal{R}_S$, then a bivalent initial state in $\mathcal{R}_S$ can be extended into an infinite run all of whose states are bivalent. We can thus obtain the basic impossibility theorem for consensus in asynchronous models.

Theorem 4.2 *Let S be a successor function such that (i) at most $t < n/2$ processes fail in runs of $\mathcal{R}_S$, (ii) $\mathcal{R}_S$ displays an arbitrary crash failure with respect to Con_0, and (iii)* *for every state x of $\mathcal{R}_S$ the set $S(x)$ is valence connected. Then $\mathcal{R}_S$ cannot satisfy all three requirements of consensus.*

Proof: Assume that $\mathcal{R}_S$ satisfies the decision and validity requirements of consensus. These requirements imply together with Lemma 3.6 that there is an initial bivalent (in $\mathcal{R}_S$) state, x^0. Applying Lemma 4.1 repeatedly with the decision property, we obtain a run r of $\mathcal{R}_S$ starting in x^0, all of whose states are bivalent. Assume by way of contradiction that $\mathcal{R}_S$ satisfies agreement. By Lemma 3.1 we have that at least $n - t > n - n/2 = n/2 > t$ processes never decide in r. Since at most $t < n/2$ processes can be faulty in r, this contradicts the assumption that $\mathcal{R}_S$ satisfies decision. ■

Notice that Lemma 3.2 yields a slightly stronger version of this theorem for systems with no finite failure.

Layering functions: A successor function S is a *layering* of a system $\mathcal{R}$ if for every S-run r^s starting in an initial state of $\mathcal{R}$ (so that $r^s(0) = r'(0)$ for some run $r' \in \mathcal{R}$), there is a run $r \in \mathcal{R}$ and a monotone mapping $\sigma : \mathbf{N} \to \mathbf{N}$ with $\sigma(0) = 0$ such that $r^s(m) = r(\sigma(m))$ for all $m \geq 0$.

Intuitively, a layering of $\mathcal{R}$ allows us to focus on "interesting states" within "interesting runs" of $\mathcal{R}$. We think of such a layering S as defining "layers" in the relevant runs of $\mathcal{R}$. The initial states of $\mathcal{R}$ are states in the initial "layer", and each application of S we think of as moving to the next layer. The next lemma says that if Theorem 4.2 applies and $\mathcal{R}_S$ cannot satisfy the consensus requirements, neither can $\mathcal{R}$.

Lemma 4.3 *Let $\mathcal{R}$ be a system for consensus and let S be a layering of $\mathcal{R}$. Then $\mathcal{R}_S$ is a system for consensus. Moreover, for each* req $\in$ *{decision, agreement, validity}, if all runs of $\mathcal{R}$ satisfy* req, *then so do all runs of $\mathcal{R}_S$.*

5 Impossibility results

Having set the stage, we can now consider applications of layering to the analysis of consensus in different models. Throughout the paper, we will focus on deterministic protocols.

We start with an impossibility for a *single mobile failure* in the synchronous model [24]. We are talking about the standard synchronous model, except that, in every round there can be at most one process some of whose messages are lost. We use the term *mobile* because the identity of the process whose messages may be lost can change from one round to the next. In this model, we can represent the environment's action at a state by a pair (j, G), with $G \subseteq \{1, \ldots, n\}$. This action means that all messages sent in the upcoming round by process j to processes in G are lost.

With respect to a given protocol, we denote by $x(j, G)$ the state that results from x when the processes follow the protocol, and the environment performs the action (j, G).[3] Notice that the environment here has sufficient power to silence a single process forever from any state of the computation. We define $\mathsf{Faulty}(i, r)$ to hold in this model exactly if there is a finite k such that i is silenced in all rounds $\ell \geq k$ of r. We denote this model by M^{mf}.

Fix a protocol $\mathcal{A}$ for the processes and let $\mathcal{R}(\mathcal{A}, M^{mf})$ be the system consisting of all runs of $\mathcal{A}$ in M^{mf}. Moreover, assume that $\mathcal{R}(\mathcal{A}, M^{mf})$ satisfies the decision requirement.

[3]The processes' next state depends on their current local states and on the environment's action; it does not depend on the environment's local state. In this model, we shall therefore ignore the environment's state and think of it as being constant.

We now define a layering function for $\mathcal{R}(\mathcal{A}, M^{mf})$. For $i = 1, \ldots, n$, let $[k]$ denote the set $\{1, \ldots, k\}$. In addition, for notational convenience we denote the empty set by $[0]$. For every state x in $\mathcal{R}(\mathcal{A}, M^{mf})$, define

$$S_1(x) = \{x(j, [k]) : 1 \leq j \leq n, \ 0 \leq k \leq n\}.$$

Thus, $S_1(x)$ contains a successor of x for every environment action of the form $(j, [k])$. Let $\mathcal{R}_{S_1}$ be the corresponding set of S_1-runs.

Lemma 5.1 *(i) S_1 is a layering of $\mathcal{R}(\mathcal{A}, M^{mf})$;*

(ii) $\mathcal{R}_{S_1}$ displays an arbitrary crash failure with respect to every subset X of its states; and

(iii) for every state x of $\mathcal{R}_{S_1}$, the set $S_1(x)$ is valence connected.

Proof: Parts (i) and (ii) are straightforward. We sketch (iii). For every j, j' we have that $x(j, [0]) = x(j', [0])$ and hence also $x(j, [0]) \sim_s x(j', [0])$. Moreover, for every $k < n$ we have that $x(j, [k]) \sim_s x(j, [k+1])$, because the two states differ only in the state of process $k + 1$. It follows that $S_1(x)$ is similarity connected for all x. That $S_1(x)$ is valence connected now follows from (ii) by Lemma 3.5. ■

Given Lemma 4.3 and Lemma 5.1(iii), Theorem 4.2 now yields

Corollary 5.2 *No protocol solves the consensus problem in the single mobile failure model.*

This result is a simple corollary of a theorem of Santoro and Widmayer in [24]. Their proof is the only one we have found that uses a bivalence-based argument in the style of Fischer et al. [14] in the synchronous context.

5.1 Impossibility in Asynchronous Models

The same outline of the impossibility proof for M^{mf} can be used in the two typical models of asynchronous systems with a single crash failure [14, 20]: message passing and shared memory. In these models, "slow" behavior of processes can be used to imitate the omitting behavior in M^{mf}. The small but crucial difference now will be that, in the asynchronous model, delayed messages will nevertheless need to eventually be delivered (or, similarly, a slow processes that is about to write a variable will ultimately write the value). In the synchronous model M^{mf}, the lost messages are gone forever. Hence, to perform a careful analysis of the round by round evolution, we will need to consider as part of the state (i.e., in the environment's local state) the status of the messages in transit or, similarly, of the current values of shared variables. In this sense, we are going slightly beyond the scope of most of the recent work on topological approaches.

Shared memory: the synchronic layering

We shall now consider the standard (see for example [22, 20]) shared-memory model M^{rw} in which variables are single-writer multiple-reader. For ease of exposition, we shall ignore the decision and other local actions, as their role is obvious, and consider only read and write actions explicitly. The shared variables are assumed to be part of the environment's local state. A process crash results in the process refraining from reading or writing from the state in which it crashes on. At most one process can crash in a given run.

We define a *local phase* for process i to be a sequence of actions performed by i with the following form: At most one write_i action, followed by a maximal sequence of $\mathsf{read}_i(V_j)$ actions in which no variable is read more than once. For every run of M^{rw} and every process i, there is a way to divide i's local history in r into a sequence of such local phases. Moreover, there is only one way to do so. Intuitively, a local phase can be viewed as an analogue of what a single process undergoes in one round in the synchronous model. We will strengthen the analogy, and define a successor function S^{rw} for M^{rw} that will mimic the layering function S_1 very closely. We restrict attention to runs in which the processes proceed in *virtual rounds*, in each of which all but at most one will perform a local phase. A round will consist of four stages:

$$\mathsf{W}_1, \quad \mathsf{R}_1, \quad \mathsf{W}_2, \quad \mathsf{R}_2$$

Reads and writes are instantaneous, in the sense that a process performing a $\mathsf{read}_i(V_j)$ operation will receive the latest value written into V_j by the period in which it is being read. By varying when in a round processes will write and when they read, we can imitate (at least temporarily) the loss (or omission) of messages in a given round. More formally, we assume that the environment has actions of two types:[4] (j, A), and (j, k) where $1 \le j \le n$ is a process name, and $0 \le k \le n$. The process j specified in the environment's action is considered the *slow* process in the layer, and the others we call *proper*. When the environment's action is (j, A) the proper processes all write their values in phase W_1 and read in phase R_1, while process j neither writes nor reads (the A stands for *absent*). When the environment's action is (j, k) the proper processes all write their values in phase W_1, while j writes in phase W_2. The proper processes $i \le k$ read in phase R_1, while process j and the proper processes $i' > k$ read in phase R_2. We remark that the state resulting from the action $(j, 0)$ applied in a state x depends on x but is independent of j, since every process performs writes based on its local state in x, and all reads occur after all writes are completed. With respect to a fixed deterministic protocol $\mathcal{A}$ for M^{rw}, we define the successor function S^{rw} by:

$$\begin{aligned} S^{rw}(x) \quad = \quad & \{x(j,k) : 1 \le j \le n,\ 0 \le k \le n\} \cup \\ & \{x(j,A) : 1 \le j \le n\} \end{aligned}$$

Whereas in the synchronous case, the states in $S_1(x)$ were indeed immediate successors of x according to the model M^{mf}, this is clearly not the case for S^{rw}. In fact, S^{rw} has an important property which will prove very useful: Every S^{rw}-run is *fair*, in the sense that all processes except at most one perform actions infinitely often. As a result, given a protocol $\mathcal{A}$ satisfying decision, S^{rw} will generate a layering of $\mathcal{R}(\mathcal{A}, M^{rw})$. We will hence avoid some of the trouble involved in proving liveness in FLP-like proofs here. We can now show:

Lemma 5.3 *The analogue of Lemma 5.1 holds for S^{rw} and M^{rw}.*

Proof: Part (i) and (ii) are again straightforward. The proof of part (iii), stating that $S^{rw}(x)$ is valence connected, proceeds in two steps. In the first, the same proof as applied for Lemma 5.1 shows that the subset Y of $S^{rw}(x)$ consisting of the states $x(j, k)$ with $k \neq A$ is valence connected. As before, Y is similarity connected, and because of part (ii) it is valence connected by Lemma 3.5. We complete the argument by showing that every state in $S^{rw}(x) \setminus Y$ has a shared valence with a state in Y, and we thus obtain that $S^{rw}(x)$ is valence connected. Specifically, we will show that $x(j, n) \sim_v x(j, A)$ for all j. In $x(j, n)$ process j had a chance to write into V_j (although nobody managed to read it yet) in the round following x, while in $x(j, A)$ it did not. It is not hard to see, however, that $y = x(j, n)(j, A)$ and $y' = x(j, A)(j, 0)$ agree modulo j. The only value written by j after x (if at all) is the same in both cases, and can be seen by all processes only in the second round. Thus, the values of all shared variables and all local states other than j's are the same, and $y \sim_s y'$. By part (ii) and Lemma 3.3 we obtain that $y \sim_v y'$, and hence $x(j, n) \sim_v x(j, A)$ and we are done. ∎

As for M^{mf}, Theorem 4.2, Lemma 4.3 and Lemma 5.1(iii) now yield the impossibility result [20]:

Corollary 5.4 *No protocol solves the consensus problem in the asynchronous r/w shared memory model while tolerating one crash failure.*

We remark that a completely analogous impossibility proof can be given for asynchronous message passing as well. The structure of the layering function, and the reasoning underlying the results remain unchanged.

Our development leading to Corollary 5.4 can reasonably be viewed as reproving the impossibility result for M^{rw} given in [20] by Loui and Abu-Amara. Indeed, we hope the reader finds this version to be somewhat more concise and perhaps easier to digest. We believe, however, that our result can be interpreted as saying quite a bit more. Technically, we have shown that consensus is unsolvable even in the submodel defined by S^{rw}. This is a model that is very close to being synchronous: In every round, at least $n-1$ processes get to write their variables and read at least $n-1$ newly written variables. By employing a full-information protocol, for example, at least $n-1$ processes will know what the round number is, and will have a view almost identical to what they would have in the synchronous case. So we have in fact shown that even with the very restricted degree of asynchrony inherent in the model defined by the actions in S^{rw}, consensus is impossible. This is perhaps the strongest explicit version so far of an FLP-like impossibility theorem. The same comments will apply in the asynchronous message passing case, where the model defined by the analogous layering function is even closer to the synchronous models that are popular in the literature.

Message passing: the permutation layering

We have just used a layering function for asynchronous systems in which a layer very closely resembles a round in the synchronous model. To illustrate the flexibility of the layering approach, we now use a very different layering, called the *permutation* layering, to present a very simple FLP-style impossibility proof. As for the synchronic layering, essentially the same proof will work in both the shared memory and the message passing models. We shall present this proof with respect to the asynchronous message passing model [14], to provide a direct and simple proof in this model too. The permutation layering is inspired by wait-free immediate snapshot executions in shared memory [5, 25, 4], and provides an analogue of these executions for message passing.[5]

[4]Notice that we are making no simplifying assumptions regarding the form of the protocols used; only the actions of the environment, or the scheduler, are being restricted.

[5]We are not aware of an analogue of immediate snapshots that has previously been suggested for message passing.

In this case, in a local phase for process i, first all outstanding messages that have been sent to i are delivered, and then process i sends messages (according to its protocol) to a maximal set of distinct destinations. (That is, in a single local phase process i sends at most one message to every other process.) Roughly speaking, rather than imitating a synchronous round, we will now have virtual rounds in which processes perform their local phases one, or infrequently two, at a time. We consider environment actions (consisting of scheduling sequences) of three types:

- $[p_1, \ldots, p_n]$,
- $[p_1, \ldots, p_{n-1}]$, and
- $[p_1, \ldots, p_{k-1}, \{p_k, p_{k+1}\}, p_{k+2}, \ldots, p_n]$ with $k < n$.

In all cases, the process names p_i in an action are pairwise distinct elements of $\{1, \ldots, n\}$. Actions of the first and second type specify a linear order in which processes are to perform local phases. An action of the third type is similar, except that processes p_k and p_{k+1} perform their local phases concurrently: first both of them receive their incoming messages, and each of them sends his messages only after the other has received its current phase messages. Fix a protocol $\mathcal{A}$ satisfying the decision requirement. We define $S^{per}(x)$ to be all states obtainable from x via one of these three actions. We call actions of the first and third type *full*, because they involve every process taking steps. As in the case of S^{rw}, we now have that every S^{per} run has all but at most one process moving infinitely often. Moreover, $R_{S^{per}}$ clearly displays an arbitrary crash failure wrt all subsets X. The proof that $S^{per}(x)$ is valence connected is slightly simpler than before. It is easy to check that

$$\begin{aligned} x[p_1, \ldots, p_n] &\sim_s x[p_1, \ldots, \{p_k, p_{k+1}\}, \ldots, p_n] \\ &\sim_s x[p_1, \ldots, p_{k+1}, p_k, p_{k+2} \ldots, p_n], \end{aligned}$$

which implies that two states that result from performing at x actions defined by permutations that differ in a single transposition are similarity connected. By transitivity of connectivity and the fact that transpositions span all permutations over $\{1, \ldots, n\}$, we obtain that the set of successors of x obtained from x via a full action is similarity connected (and hence valence connected). Finally, we show that $x[p_1, \ldots, p_{n-1}, p_n] \sim_v x[p_1, \ldots, p_{n-1}]$ by demonstrating that they have a common successor y. Whatever valence y has is a shared valence for both states. The state y is given by:

$$\begin{aligned} y &= x[p_1, \ldots, p_{n-1}, p_n][p_1, p_2, \ldots, p_{n-1}] \\ &= x[p_1, \ldots, p_{n-1}][p_n, p_1, p_2, \ldots, p_{n-1}]. \end{aligned}$$

Equality holds because in both cases the exact same sequence of basic actions happens in the two rounds following x. Here the FLP diamond argument is reduced to its bare minimum. Notice that we do not have $x[p_1, \ldots, p_{n-1}, p_n] \sim_s x[p_1, \ldots, p_{n-1}]$ because the two states can differ both in the state of p_n *and* in the state of the environment: in the former case p_n has sent messages (in M^{rw} it could have written into V_n), while in the latter p_n has not done so. This pinpoints the reason why the diamond argument, and indeed reasoning about valence, are useful in the asynchronous models.

6 The Synchronous Lower Bound

The analysis we performed for the mobile failure model M^{mf} in the synchronous case should, intuitively, apply equally well to the standard t-resilient case in the synchronous model, where there is a bound of t on the total number of processes who may fail in the run, and a process some of whose messages are lost is considered faulty. The well-known lower bound for this case (due originally to [13, 10]) states that every consensus protocol must require at least $t+1$ rounds in its worst case run. In this model, the environment can use one failure in every round for t to simulate a prefix of an S_1-run for t rounds. One could hope to show that there will exist a bivalent state at the end of round t, and thus derive the $t+1$-round lower bound directly from our analysis for M^{mf}. A close inspection, however, shows that things are not *that* simple. There will typically not need to be a bivalent state at the end of round t. But the essence of this idea still works.

We assume $1 \leq t \leq n-2$, and hence $n \geq 3$. About the failure model we assume (i) that in the first round in which a process fails the environment can block the delivery of an arbitrary subset of its messages, (ii) that the environment can silence a faulty processor forever in all rounds after the first one in which it fails, and (iii) the environment's local state keeps track of the processes that have failed. For this model, given a t-resilient consensus protocol $\mathcal{A}$ we define a layering function S^t by

$$S^t(x) = \begin{cases} S_1(x) & \text{if fewer than } t \text{ are failed at } x\text{, and} \\ x(1, [0]) & \text{otherwise.} \end{cases}$$

Recall that $x(1, [0])$ is a successor of x in which no process fails. Thus, in an S^t layer at most one process performs an omitting failure (and then is recorded as having failed and is silenced forever after) so long as fewer than t processes have already failed. Once t processes fail, no more failures happen. Thus, clearly, $\mathcal{R}_{S^t}$ displays an arbitrary crash failure with respect to every set X of states in which fewer than t failures are recorded. And as in the case of M^{mf}, it is straightforward to verify that S^t is a layering function for $\mathcal{A}$ in a synchronous model of this type. The valence connectedness property of S_1 stated in Lemma 5.1(iii) holds in this model as well. We now have:

Lemma 6.1 *Let x^0 be a bivalent state of $\mathcal{R}_{S^t}$ in which no more than f processes are failed. Then there is an S^t-execution $x^0, x^1, \ldots, x^{t-f-1}$ of $\mathcal{A}$, such that x^{t-f-1} is bivalent and no more than $t-1$ processes are failed in x^{t-f-1}.*

Proof: We prove by induction on m, for $0 \leq m \leq t-f-1$, that an execution of the desired form exists, with x^m bivalent and where no more than $m+f$ processes are failed in x^m. The basis $m=0$ holds by assumption. Assume inductively that the claim holds for $m < t-f-1$. Thus, we have that $m+f < t-1$ processes are failed in x^m. Recall that if fewer than $t-1$ processes are failed in a state x then (i) $S^t(x) = S_1(x)$ and (ii) $\mathcal{R}_{S^t}$ displays an arbitrary crash failure with respect to $S^t(x)$. Thus, Lemma 5.1 (in its version for this model, and assuming $n \geq 3$) implies that $S^t(x^m)$ is valence connected. It follows from Lemma 4.1 that there is a bivalent state $x^{m+1} \in S^t(x^m)$. By definition of S^t, the number of failed processes in x^{m+1} is at most $f+m+1$. ∎

The existence of a bivalent initial state with $f=0$ failed processes, Lemma 6.1 immediately implies the existence of a bivalent state x^{t-1} at the end of round $t-1$. By Lemma 3.1, this gives us a t-round lower bound for consensus. The true $(t+1)$-round lower bound is obtained by showing that two rounds are still necessary after a bivalent state:

Lemma 6.2 *Assume that $\mathcal{A}$ is a protocol for consensus. Let $\hat{x}$ be a bivalent state of $\mathcal{R}_{S^t}$. If $\hat{x}$ is bivalent, then there is a state $y \in S^t(\hat{x})$ in which at least one non-failed process has not decided.*

Proof: Notice that a state x with t failed processes must be univalent, since there is a unique infinite S^t extension starting at x. Hence, to be bivalent, the state $\hat{x}$ can have no more than f, $f \leq t-1$, failed processes. Since $f \leq t-1$, we have from the definition of S^t that $S^t(\hat{x}) = S_1(\hat{x})$, and the proof of Lemma 5.1 (in its version for this model, and assuming $n \geq 3$) states that $S^t(\hat{x}) = S_1(\hat{x})$ is similarity connected. Since $\hat{x}$ is bivalent, there are states $y^0, y^1 \in S^t(\hat{x})$ such that y^0 is 0-valent and y^1 is 1-valent. The similarity connectivity of $S^t(\hat{x})$ implies the existence of states $z^0, z^1 \in S^t(\hat{x})$ (not necessarily distinct) satisfying $z^0 \sim_s z^1$ and that are 0- and 1-valent, respectively. Recall that all states of $\mathcal{R}_{S^t}$ have at most t faulty processes. Since $t \leq n-2$ and z^0 and z^1 agree modulo j for some j, it follows that there is at least one process $i \neq j$ that is not failed in both states such that $z^0_i = z^1_i$. Assume for contradiction that every non failed process is decided in both z^0 and z^1. In particular, i is decided, say with value v. Agreement implies that in both states, every nonfaulty process decides v, and hence both z^0 and z^1 are v-univalent. ■

We can now put the two results together and obtain the desired lower bound:

Corollary 6.3 *Every t-resilient protocol for consensus in the synchronous model where faulty processes can crash has a run in which decision requires at least $t+1$ rounds.*

This result was first proved for crash failures by Dolev and Strong [10], and the latest version of the proof is in [11]. Our proof here is the first one we are aware of that is in the style and spirit of the impossibility proofs for the asynchronous case. Moreover, we feel that it is even simpler than the one of [11]. In addition to generalizing the lower bound for t-resilient consensus, we feel that our proof provides further insight into the structure of consensus protocols in this model. Let us briefly consider an example. It is well-known [23] that there are t-resilient consensus protocols that are guaranteed to decide in precisely $t+1$ rounds. Thus, the worst-case lower bound of Dolev and Strong is tight. Let us call a protocol in which consensus is always reached in at most $t+1$ rounds *fast*. We can now show

Lemma 6.4 *Let $\mathcal{A}$ be a fast t-resilient consensus protocol. For every execution $x^0, x^1, \ldots, x^k, x^{k+1}$ of $\mathcal{A}$, if at most k processes have failed by x^k, and the $k+1$st round is failure-free, then x^{k+1} must be univalent.*

Proof: By assumption, only k processes have failed by x^{k+1}. If x^{k+1} is bivalent, then by Lemma 6.1 it can be extended to a run with a bivalent state x^t at the end of t rounds. By Lemma 6.2, two more rounds are necessary for agreement in the worst case, contradicting the assumption that $\mathcal{A}$ is fast. ■

Clearly, Lemma 6.1 also partially describes the situation in runs in which potentially more than one process can crash in a given round. It matches the upper (and lower) bounds given in [11], which show roughly that if in some execution $k+w$ crashes are detected by the end of round k, then agreement can be secured by the end of round $t+1-w$. Hence, by allowing $k+w$ crashes by the end of round k, the environment has essentially "wasted" w faults in its quest to delay agreement. Lemma 6.1 guarantees that the environment has not lost more than w rounds in this case.

7 Decision Problems

We briefly describe how the techniques developed for consensus apply also to general decision problems. Due to lack of space, we defer a more complete discussion to the full paper.

For the purpose of this paper, a *vertex* is a pair $\langle i, v \rangle$, consisting of a process id $i \in \{1, \ldots, n\}$ and a value v from some range V. A *simplex* is a set of vertices in which all process id's are distinct. Thus, a simplex can consist of at most n vertices. A *k-size-simplex* is one that contains k vertices. In a given run, an *input simplex* is a simplex describing the initial inputs of the processes, and an *output simplex* is one that describes the decisions taken by a set of processes. A *complex* is a set of simplexes that is closed under containment. In an *n-size-complex* the maximal simplexes have n elements.

The consensus problem is an example of a *decision problem* (e.g. [7]). Roughly, these are problems where processes start with input values, communicate with each other, and decide on output values according to the problem specification. More precisely, a *decision problem $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$* consists of an *input complex, $\mathcal{I}$*, an *output complex, $\mathcal{O}$*, and a mapping $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$. The environment is in the same local state in all initial states, and we denote this set of initial states for $\mathcal{D}$ by $\mathcal{D}_0$. The decision problem for a model M is to design a protocol $\mathcal{A}$ for M, such that every run r of $\mathcal{A}$ in M satisfies two conditions: *Decision,* requiring that every nonfaulty process eventually decides, and *Validity,* requiring that the decisions made in a run starting in an input simplex s form a simplex in $\Delta(s)$. Thus, to solve a decision problem $\mathcal{D}$ it is enough to solve a *subproblem, $\mathcal{D}' = \langle \mathcal{I}, \mathcal{O}, \Delta' \rangle$*, where $\Delta'(s) \subseteq \Delta(s)$ for every input simplex s.

Our purpose is to relate the solvability of general decision problems to that of consensus. To this end, we say that a pair $\mathcal{O}_1, \mathcal{O}_2$ of n-size complexes is a *covering* of a set of runs R if (i) every decided output simplex of a run of R is in one or both of the complexes, and (ii) each of $\mathcal{O}_0, \mathcal{O}_1$ contains at least one decided output simplex of a run of R. Intuitively, the two parts of a covering can be thought of as defining "decision values" to be chosen by the run. This is captured as follows.

Generalized Valence. With respect to a set of runs R and a covering $\mathcal{O}_0, \mathcal{O}_1$ of R, we define a state x in R to be *v-valent*, $v \in \{0, 1\}$, if there is a run r of R extending x such that the simplex describing the decisions made by nonfaulty processes in r is a simplex of $\mathcal{O}_v$. The notions of *univalence* and *bivalence* are defined based on v-valence as in the case of consensus.

We remark that the sets R that will be of interest to us will typically have a particular form: They will consists of the set of all runs of a system $\mathcal{R}$ in which states of a set X appear. We denote such a set by $\mathcal{R}(X)$.

Generalized Connectivity. The notion of *similarity* between states is unchanged, and for the generalized notion of valence, we define *shared valence* with respect to a covering $\mathcal{O}_0, \mathcal{O}_1$ as in Definition 3.1, and maintain the corresponding connectivity definitions. It turns out that we may be interested in many possible coverings. Given a system $\mathcal{R}$, we say that a set X of states of $\mathcal{R}$ is *always valence connected* if it is valence connected w.r.t. every covering $\mathcal{O}_0, \mathcal{O}_1$ of $\mathcal{R}(X)$.

Always valence connectedness will now play a role analogous to that played by valence connectedness in the case of consensus. Variants of Lemmas 3.3 and 3.4 in which *valence connected* is replaced by *always valence connected* still hold, and their proofs do not change, and hence so do the corresponding versions of Lemma 3.5 and Lemma 4.1.

The following is a more general version of Theorem 4.2, but the proof is almost the same.

Lemma 7.1 *Assume that (i) $\mathcal{R}_S$ satisfies decision and (ii) for every state x of $\mathcal{R}_S$ the set $S(x)$ is always valence connected. Let I be a similarity connected set of initial states such that $\mathcal{R}_S$ displays an arbitrary crash failure with respect to I, and let $\mathcal{O}_0, \mathcal{O}_1$ be a covering of $\mathcal{R}_S(I)$. Then there is a run $r^b \in \mathcal{R}_S(I)$ all of whose states are bivalent.*

Proof: Since I is similarity connected, $\mathcal{R}_S$ satisfies decision, and $\mathcal{R}_S(I)$ displays an arbitrary crash failure with respect to I, we have by Lemmas 3.3 and 3.5 that I is always valence connected. In particular, I is valence connected with respect to $\mathcal{O}_0, \mathcal{O}_1$. Since $\mathcal{O}_0, \mathcal{O}_1$ is a covering of $\mathcal{R}_S(I)$, there exist at least one 0-valent and one 1-valent state in I. It follows from Lemma 3.4 that there is a state $x^0 \in I$ that is bivalent with respect to $\mathcal{O}_0, \mathcal{O}_1$. Assume inductively that we have constructed an S-execution $x^0, \ldots, x^k$ with $x^0 \in I$ and where all states are bivalent. We claim that $\mathcal{O}_0, \mathcal{O}_1$ is a covering of $\mathcal{R}_S(S(x^k))$. Clearly, $\mathcal{R}_S(S(x^k)) \subseteq \mathcal{R}_S(I)$, so that every decided output simplex of $\mathcal{R}_S(S(x^k))$ is one of $\mathcal{R}_S(I)$ as well, and hence appears in $\mathcal{O}_0 \cup \mathcal{O}_1$. Moreover, because x^k is bivalent, each of $\mathcal{O}_0, \mathcal{O}_1$ contains at least one decided output simplex of a run of $\mathcal{R}_S(S(x^k))$ and the claim follows. By assumption, $S(x^k)$ is always valence connected, and thus valence connected with respect to $\mathcal{O}_0, \mathcal{O}_1$. By Lemma 4.1 there is a state $y \in S(x^k)$ that is bivalent with respect to $\mathcal{O}_0, \mathcal{O}_1$. Define $x^{k+1} = y$ to be the next state in the execution. The run $r^b = x^0, \ldots, x^k, x^{k+1}, \ldots$ obtained by this construction is a run of $\mathcal{R}_S$ all of whose states are bivalent. ■

Notice that if $\mathcal{R}_S$ satisfies decision, then the simplex describing the decided outputs in the run r^b constructed in Lemma 7.1 would be in the intersection $\mathcal{O}_0 \cap \mathcal{O}_1$. There is a precise sense in which this shows that the covering does not divide the set of output simplexes into two disconnected parts. Since the conditions of the lemma are met in the presence of a potential crash and similarity connectedness of the inputs, the lemma can be used to characterize the set of decision problems solvable in many models of interest. We start with a slightly technical characterization of a necessary condition for solvability.

An n-size-complex $\mathcal{C}$ is *k-thick-connected* if for every pair s_1, s_2 of n-size-simplexes in $\mathcal{C}$ there exists a sequence of n-size-simplexes leading from one to the other in which every two consecutive n-size-simplexes contain in their intersection an $(n-k)$-size-simplex. For the purposes of the next theorem, for a set I of initial states, we define $\mathcal{C}_\Delta(I)$ to be the complex generated by the set $\{\Delta(s) : s$ is the input simplex of a state in $I\}$. Finally, we say that a decision problem $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is *k-thick connected* if there is a subproblem $\mathcal{D}' = \langle \mathcal{I}, \mathcal{O}, \Delta' \rangle$ of $\mathcal{D}$ such that for every similarity connected set $I \subseteq \mathcal{D}_0$ of initial states of $\mathcal{D}$, the complex $\mathcal{C}_{\Delta'}(I)$ is k-thick-connected. We can now show

Theorem 7.2 *Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ and let S and $\mathcal{R}_S$ satisfy (i) $\mathcal{R}_S$ displays an arbitrary crash failure with respect to $\mathcal{D}_0$, (ii) no run of $\mathcal{R}_S$ contains more than one faulty process, and (iii) for every state x of $\mathcal{R}_S$ the set $S(x)$ is always valence connected. If $\mathcal{R}_S$ satisfies decision and validity then $\mathcal{D}$ is 1-thick connected.*

This theorem provides a necessary condition for solvability for each one of the models in which we showed consensus not to be solvable 1-resiliently (the single mobile failure model, the r/w shared memory model and the asynchronous message passing model, as well as the submodels defined by the layerings we provided for them): A decision problem $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ is solvable 1-resiliently exactly if $\mathcal{D}$ is 1-thick connected. This condition essentially coincides with the one in [7]. Since in [7] the condition was shown to be also sufficient, then it is also sufficient in the submodels that we have presented. That is, in all these models, the same problems are solvable 1-resiliently. Formally, we are able to show

Corollary 7.3 *For each of the following models, a decision problem $\mathcal{D}$ is solvable if and only if it is 1-thick connected:*

- *1-resilient r/w shared memory M^{rw},*
- *1-resilient message passing,*
- *the submodels of the above defined by the synchronic and the permutation layerings, and*
- *the single mobile failure model M^{mf}.*

In the full paper we use the same techniques to extend the equivalence to snapshot shared memory [2], iterated immediate snapshot [6], and related models.

The Synchronous Model

We now consider decision problems in the t-resilient synchronous model of computation of Section 6. It is well-known that consensus and a large class of decision problems are solvable in $t+1$ rounds in the synchronous model. What we have seen in Section 6 is that at least in the case of consensus, there is a close connection between the first t rounds in the synchronous model and the popular asynchronous models. Here we present a necessary condition for solvability by t rounds. Although the condition is similar to the one for the asynchronous systems, it is stronger: There are problems that are solvable 1-resiliently in the asynchronous models we considered, that are not solvable in t rounds of the synchronous model (cf. [8]).

Fix an arbitrary algorithm $\mathcal{A}$ that runs for at most t rounds, and the layering function S^t. We have the generalized version of Lemmas 6.1 and 6.2:

Lemma 7.4 *Let I be a similarity connected set of initial states, and let $\mathcal{O}_0, \mathcal{O}_1$ be a covering of $\mathcal{R}_{S^t}(I)$. If $\mathcal{R}_{S^t}$ satisfies decision then there is a run r of $\mathcal{R}_{S^t}$ with prefix $x^0, x^1, \ldots, x^{t-1}, x^t$, where $x^0 \in I$ and for every m, $0 \le m \le t$ the state x^m is bivalent and no more than m processes are failed in x^m.*

We can now obtain a necessary condition for solvability by t rounds, as in Theorem 7.2:

Lemma 7.5 *Assume $\mathcal{R}_{S^t}$ satisfies decision and validity w.r.t. $\mathcal{D}$, and that in all runs of $\mathcal{R}_{S^t}$ all decisions are made within t rounds. Then $\mathcal{D}$ is t-thick connected.*

This necessary condition is not sufficient for solvability by t rounds. We now prove a stronger necessary condition, which proves that there are decision problems that are solvable 1-resiliently in the asynchronous models and are not solvable in t rounds of the synchronous model. The additional condition (in the style of [8]) bounds the diameter of $\Delta'(s)$, for any input simplex s. Formally, we define the *s-diameter* of a set of states X to be the diameter of the graph $(X, \sim_s)$ induced by the similarity relation on X.

Lemma 7.6 *Let S be a successor function such that (i) $\mathcal{R}_S$ displays an arbitrary crash failure with respect to a set of states X, (ii) X is s-connected, and (iii) for every state $x \in X$ the set $S(x)$ is s-connected. Then $S(X)$ is s-connected. Moreover, if the s-diameter of X is d_X and the s-diameter of $S(x)$, $x \in X$, is at most d_Y, then the s-diameter of $S(X)$ is at most $d_X d_Y + d_X + d_Y$.*

For the next theorem we apply Lemma 7.6 repeatedly with S^t. We can show that $d_Y^m = 2(n-m)$ is an upper bound on the diameter of $S^t(x)$ for states x at the end of round m. Denote by d_X^m the diameter of the set of states at the end of round m ($d_X^0 = d(I)$). Then $d_X^{m+1} = d_X^m d_Y^m + d_X^m + d_Y^m$, using Lemma 7.6 and the definition of S^t. We get the desired necessary condition.

Theorem 7.7 *Let $\mathcal{D} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ and assume $\mathcal{R}_{S^t}$ satisfies decision and validity, and that all decisions in $\mathcal{R}_{S^t}$ are made within the first t rounds. Then there is a subproblem $\mathcal{D}' = \langle \mathcal{I}, \mathcal{O}, \Delta' \rangle$ of $\mathcal{D}$ such that for every similarity connected set $I \subseteq \mathcal{D}_0$ of initial states of $\mathcal{D}$ we have (i) the complex $\mathcal{C}_{\Delta'}(I)$ is k-thick-connected, and (ii) the diameter of $\mathcal{C}_{\Delta'}(I)$ is at most d_X^t.*

Acknowledgments: We would like to thank Juan Garay for pointing to us the mobile failure model and the related reference [24].

References

[1] Baruch Awerbuch, "Complexity of Network Synchronization," *Journal of the ACM,* Vol. 32, No. 4, Oct. 1985, pp. 804-823.

[2] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, N. Shavit, "Atomic snapshots of shared memory," *Journal of the ACM,* Vol. 40, No. 4, (September 1993), pp. 873-890.

[3] H. Attiya, A. Bar-Noy and D. Dolev, "Sharing Memory Robustly in Message-Passing Systems," *Journal of the ACM,* Vol. 42, No. 1 (January 1995), pp. 124-142.

[4] Hagit Attiya and Sergio Rajsbaum, "The Combinatorial Structure of Wait-free Solvable Tasks," *10th International Workshop on Distributed Algorithms (WDAG),* October 1996, (O. Babaoglu and K. Marzullo, Eds.), pp. 321-343. Lecture Notes in Computer Science #1151, Springer-Verlag.

[5] E. Borowsky and E. Gafni, "Generalized FLP impossibility result for t-resilient asynchronous computations," in *Proceedings of the 1993 ACM Symposium on Theory of Computing,* May 1993.

[6] E. Borowsky and E. Gafni, "A simple algorithmically reasoned characterization of wait-free computations," in *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing,* pages 189-198, 1997.

[7] O. Biran, S. Moran, S. Zaks, "A combinatorial characterization of the distributed 1-solvable tasks," *Journal of Algorithms,* Vol. 11 (1990), pp. 420-440.

[8] O. Biran, S. Moran and S. Zaks, "Tight Bounds on the Round Complexity of Distributed 1-solvable Tasks," *Theoretical Computer Science,* Vol. 145 (1995) pp. 271-290.

[9] T. Chandra and S. Toueg, "Unreliable failure detectors for asynchronous systems,". in *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing,* pages 257-272, 1991.

[10] D. Dolev, H.R. Strong, " Polynomial algorithms for multiple processor agreement," in *Proceedings of the 14th Annual ACM Symposium on Theory of Computing,* pp. 401-407, May 1982.

[11] C. Dwork, Y. Moses, "Knowledge and common knowledge in a byzantine environment: crash failures," *Information and Computation,* vol. 8, no. 2, pp. 156-186, October 1990.

[12] M. Fischer, "The consensus problem in unreliable distributed systems (a brief survey)," Research Report YALE/DCS/RR-273, Yale University, Department of Computer Science, New Haven, Conn., June 1983.

[13] M.J. Fischer, N.A. Lynch, "A lower bound for the time to assure interactive consistency," *Information Processing Letters,* vol. 14, no. 4, pp. 183-186, June 1982.

[14] M. Fischer, N.A. Lynch, and M.S. Paterson, "Impossibility of distributed commit with one faulty process," *Journal of the ACM,* 32(2), April 1985.

[15] E. Gafni, "Round-by-round fault detectors: unifying synchrony and asynchrony," these proceedings.

[16] M.P. Herlihy, "Wait-Free Synchronization," *ACM Transactions on Programming Languages and Systems,* vol. 13, no. 1, pp. 123-149, January 1991.

[17] M.P. Herlihy, S. Rajsbaum, M. R. Tuttle, "Unifying synchronous and asynchronous message-passing models," these proceedings.

[18] M.P. Herlihy and N. Shavit, "The asynchronous computability theorem for t-resilient tasks," In *Proceedings of the 1993 ACM Symposium on Theory of Computing,* May 1993.

[19] Gunnar Hoest and Nir Shavit, "Towards a topological characterization of asynchronous complexity," Proc. of the 16th Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, 1997, pp. 199-208.

[20] M. C. Loui and H.H. Abu-Amara, "Memory requirements for agreement among unreliable asynchronous processes," In *Parallel and Distributed Computing,* F. P. Preparata, editor, vol. 4 of *Advances in Computing Research,* pages 163-183. JAI Press, 1987.

[21] Ronit Lubitch and Shlomo Moran, "Closed schedulers: a novel technique for analyzing asynchronous protocols," *Distributed Computing,* Vol. 8, 1995, pp. 203-210.

[22] N.A. Lynch, *Distributed Algorithms,* Morgan Kaufmann Publishers, Inc. 1996.

[23] M., Pease, R. Shostak and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM,* Vol. 27, No. 2, pp. 228-234.

[24] N. Santoro and P. Widmayer, "Time is not a healer," In Proc. 6th Annual Symp. Theor. Aspects of Computer Science, Paderborn, Germany, February 1989. Springer Verlag LNCS Vol. 349 pp. 304-313.

[25] M. Saks and F. Zaharoglou, "Wait-free k-set agreement is impossible: The topology of public knowledge," In *Proceedings of the 1993 ACM Symposium on Theory of Computing,* May 1993.