T D C C

P F

Department of Computer Science, University of Crete
P.O. Box 2208 GR-714 09 Heraklion, Crete, Greece
and
Institute of Computer Science (ICS)
Foundation for Research and Technology (FORTH)
N. Plastira 100. Vassilika Vouton
GR-700 13 Heraklion, Crete, Greece
faturu@csd.uoc.gr

Bulletin of the EATCS no 106, pp. 53–77, February 2012

UNDERSTANDING NON-UNIFORM FAILURE MODELS

Petr Kuznetsov
TU Berlin/Deutsche Telekom Laboratories
petr.kuznetsov@tu-berlin.de

Abstract

Traditionally, models of fault-tolerant distributed computing assume that failures are "uniform": processes are equally probable to fail and a failure of one process does not affect reliability of the others. In real systems, however, processes may not be equally reliable. Moreover, failures may be correlated because of software or hardware features shared by subsets of processes. In this paper, we survey recent results addressing the question of what can and what cannot be computed in systems with non-identical and non-independent failures.

L'égalité sera peut-être un droit, mais aucune puissance humaine ne saura le convertir en fait.[1]

Honoré de Balzac

1 Introduction

A distributed system is a collection of computing units, called *processes*. The principal challenge of distributed computing is to devise protocols that correctly operate in the presence of failures of processes and asynchrony. A *failure model* describes the assumptions on where and when failures might occur. The classical "uniform" failure model assumes that processes fail with equal probabilities, independently of each other. This enables reasoning about the maximal number of processes that may, with a non-negligible probability, fail in any given execution of the system. It is natural to ask questions of the kind: what problems can be solved *t-resiliently*, i.e., assuming that at most *t* processes may fail. In particular,

[1] Equality may perhaps be a right, but no human power can ever turn it into a fact.

the *wait-free* ($(n-1)$-resilient, where n is the number of processes) model assumes that any subset of processes may fail.

However, in real systems, processes do not always fail in the uniform manner. Processes may be unequally reliable and prone to correlated failures. A software bug makes all processes using the same build vulnerable, a router's failure may makes all processes behind it unavailable, a successful malicious attack on a given process increases the chances to compromise processes running the same software, etc. Thus, understanding how to deal with non-uniform failures is crucial.

Adversaries. Consider a system of three processes, p, q, and r. Suppose that p is very unlikely to fail, and otherwise, all failure patterns are allowed. Since we only exclude executions in which p fails, the set of correct processes in any given execution must belong to $\{p, pq, pr, pqr\}$[2].

Now we give an example of correlated failures. Suppose that p and q share a software component x, p and r share a software component y, and q and r are built atop the same hardware platform z (Figure 1). Further, let x, y, and z be prone to failures, but suppose that it is very unlikely that two failures occur in the same execution. Hence, the possible sets of correct processes in our system are $\{pqr, p, q, r\}$.

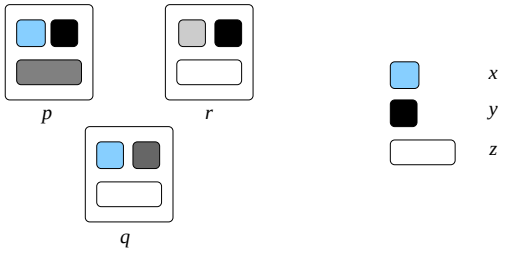

Figure 1: A system modeled by the adversary $\{pqr, p, q, r\}$: p and q share component x, p and r share component y, and q and r run atop the same hardware platform z.

The notion of a generic *adversary* introduced by Delporte et al. [9] intends to model such scenarios. An adversary $\mathcal{A}$ is defined as a set of possible correct process subsets. E.g., the *t-resilient* adversary $\mathcal{A}_{t\text{-}res}$ in a system of n processes consists of all sets of $n - t$ or more processes. We say that an execution is $\mathcal{A}$-*compliant* if the set of processes that are correct in that execution belongs to $\mathcal{A}$. Thus, an adversary $\mathcal{A}$ describes a model consisting of $\mathcal{A}$-compliant executions.

[2]For brevity, we simply write pqr when referring to the set $\{p, q, r\}$.

The formalism of adversaries [9] assumes that processes fail only by crashing, and adversaries only specify the *sets* of processes that may be correct in an execution, regardless of the timing of failures. Of course, this sorts out many kinds of possible adversarial behavior, such as malicious attacks or timing failures. However, it is probably the simplest model that still captures important features of non-uniform failures.

Distributed tasks. In this paper, we focus on a class of distributed-computing problems called *tasks*. A task can be seen as a distributed variant of a function from classical (centralized) computing: given a distributed input (an *input vector*, specifying one input value for every process) the processes are required to produce a distributed output (an *output vector*, specifying one output value for every process), such that the input and output vectors satisfy the given *task specification*.

The classical theory of computational complexity theory categorizes functions based on their inherent difficulty (e.g., with respect to solving them on a Turing machine). In the distributed setting, the difficulty in solving a task also depends on the adversary we are willing to consider. There are tasks that can be trivially solved on a Turing machine, but are not solvable in the presence of some distributed adversaries. For example, the fundamental task of *consensus*, in which the processes must agree on one of the input values, cannot be solved assuming the 1-resilient adversary $\mathcal{A}_{1\text{-}res}$ [11, 28]. More generally, the task of k-set consensus [8], where every correct process is required to output an input value so that at most k different values are output, cannot be solved in the presence of $\mathcal{A}_{k\text{-}res}$ [21, 30, 4].

Most of this paper deals with *colorless* tasks (also called convergence tasks [5]). Informally, colorless tasks allow every process to adopt an input or output value from any other participating process. Colorless tasks include consensus [11], k-set consensus [8] and simplex agreement [22].

The relative power of an adversary. This paper primarily addresses the following question. Given a task T and an adversary $\mathcal{A}$, is T solvable in the presence of $\mathcal{A}$?

Intuitively, the more sets an adversary comprises, the more executions our system may expose, and, thus, the more powerful is the adversary in "disorienting" the processes. In this sense, the *wait-free* adversary $\mathcal{A}_{wf} = \mathcal{A}_{n-1\text{-}res}$ is the most powerful adversary, since it describes the set of *all* possible executions.

In contrast, a "singleton" adversary $\mathcal{A} = \{S\}$ that consists of only one set $S \subseteq \mathcal{P}$ is very weak. For example, we can use any process in S as the "leader" that never fail. This allows us to solve consensus or implement any sequential data type [18].

But in general, there are exponentially many adversaries defined for n processes that are not related by containment. Therefore, it is difficult to say a priori which of two given adversaries is stronger.

Superset-closed adversaries. We start with recalling the model of *dependent failures* proposed by Junqueira and Marzullo [25], defined in terms of *cores* and *survivor sets*. In brief, a survivor set is a minimal subset of processes that can be the set of correct processes in some execution, and a core is a minimal set of processes that do not all fail in any execution.

We show that, in fact, the formalism of [25] describes a special class of *superset-closed* adversaries: every superset of an element of such an adversary $\mathcal{A}$ is also an element of $\mathcal{A}$. The minimal elements of $\mathcal{A}$ (no subset of which are in $\mathcal{A}$) are the survivor sets of the resulting model.

It turns out that the power of a superset-closed adversary $\mathcal{A}$ in solving colorless tasks is precisely characterized by the size of its minimal core, i.e., the minimal-cardinality set of processes that cannot all fail in any $\mathcal{A}$-compliant execution. A superset-closed adversary with minimal core size c allows for solving a colorless task T if and only if T can be solved $(c-1)$-resiliently. In particular, if $c = 1$, then any task can be solved in the presence of $\mathcal{A}$, and if $c = n$, then $\mathcal{A}$ only allows for solving wait-free solvable tasks. Thus, all superset-closed adversaries can be categorized in n classes, based on their minimal core sizes.

We present two ways of deriving this result: first, using the elements of modern topology (proposed by Herlihy and Rajsbaum [20]) and second, through shared-memory simulations (proposed by Gafni and Kuznetsov [16]).

Characterizing generic adversaries. The dependent-failure formalism of [25] is however not expressive enough to capture the task solvability in generic non-uniform failure models. It is easy to construct an adversary that has the minimal core size n but allows for solving tasks that can cannot be wait-free solved. One example is the "bimodal" adversary $\{pqr, p, q, r\}$ (Figure 1) that allows for solving 2-set consensus.

Therefore, to characterize the power of a generic adversary, we need a more sophisticated criterion than the minimal core size. Surprisingly, such a criterion, that we call *set consensus power*, is not difficult to find. Suppose that we can partition an adversary $\mathcal{A}$ into k sub-adversaries, each powerful enough to solve consensus. We conclude that $\mathcal{A}$ allows for solving k-set consensus: simply run k consensus algorithms in parallel, each assuming a distinct sub-adversary. Moreover, we show that the set consensus power of $\mathcal{A}$, defined as the minimal such number of sub-adversaries, precisely characterizes the power of $\mathcal{A}$ in solving colorless tasks.

Therefore, generic adversaries defined on n processes can still be split into n equivalence classes. Each class j consists of adversaries of set consensus power j that agree on the set of colorless tasks they allow for solving: namely, tasks that can be solved $(j-1)$-resiliently and not j-resiliently. In particular, class n contains adversaries that only allow for solving tasks that can be solved wait-free, and class 1 allows for solving consensus and, thus, any task.

Roadmap. We begin with a background section that states recalls the basics of our model and the notion of a distributed task. Then we discuss several approaches to model non-uniform failures: dependent failure model of Junqueira and Marzullo [25], adversaries of Delporte et alii [9], and asymmetric progress conditions by Imbs et alii [24].

Then we present a complete characterization of superset-closed adversaries. The result is first shown using elements of combinatorial topology [20] and then through simple shared-memory simulations [16].

We then characterize generic (not necessarily superset-closed) adversaries using the notion of set consensus power and relate it with the *disagreement power* proposed by Delporte et alii [9].

We conclude with a brief overview of open questions, primarily related to solving generic (not necessarily colorless) tasks in the presence of generic (not necessarily superset-closed) adversaries.

The results described in this paper originally appeared in [9, 14, 20, 16, 24, 31].

2 Background

In this section, we briefly state our system model and recall the notion of a distributed task and two important constructs used in this paper: Commit-Adopt and BG-simulation.

2.1 Model

We consider a system Π of n processes, $p_1, \ldots, p_n$, that communicate via reading and writing in the shared memory. We assume that the system is *asynchronous*, i.e., relative speeds of the processes are unbounded. Without loss of generality, we assume that processes share an *atomic snapshot* memory [1], where every process may update its dedicated element and take atomic snapshot of the whole memory.

A process may only fail by crashing, and otherwise it must respect the algorithm it is given. A *correct* process never crashes.

2.2 Tasks

In this paper, we focus on a specific class of distributed computing problems, called *tasks* [22]. In a distributed task [22], every participating process starts with a unique input value and, after the computation, is expected to return a unique output value, so that the inputs and the outputs across the processes satisfy certain properties. More precisely, a *task* is defined through a set $\mathcal{I}$ of input vectors (one input value for each process), a set O of output vectors (one output value for each process), and a total relation $\Delta : \mathcal{I} \mapsto 2^O$ that associates each input vector with a set of possible output vectors. An input $\perp$ denotes a *not participating* process and an output value $\perp$ denotes an *undecided* process.

For example, in the task of *k-set consensus*, input values are in $\{\perp, 0, \ldots, k\}$, output values are in $\{\perp, 0, \ldots, k\}$, and for each input vector I and output vector O, $(I, O) \in \Delta$ if the set of non-$\perp$ values in O is a subset of values in I of size at most k. The special case of 1-set consensus is called *consensus* [11].

We assume that every process runs a *full-information* protocol: initially it writes its input value and then alternates between taking snapshots of the memory and writing back the result of its latest snapshots. After a certain number of such asynchronous rounds, a process may gather enough state to *decide*, i.e., i.e., to produce an irrevocable non-$\perp$ output value.

In *colorless* task (also called *convergence* tasks [5]) processes are free to use each others' input and output values, so the task can be defined in terms of input and output *sets* instead of vectors.[3] The k-set consensus task is colorless.

Note that to solve a colorless task, it is sufficient to find a protocol (a decision function) that allows just one process to decide. Indeed, if such a protocol exists, we can simply convert it into a protocol that allows every correct process to decide: every process simply applies the decision function to the observed state of any other process and adopts the decision.

2.3 The Commit-Adopt protocol

One tool extensively used in this paper is the *commit-adopt* abstraction (CA) [12]. CA exports one operation *propose*(v) that returns $(commit, v')$ or $(adopt, v')$, for $v', v \in V$, and guarantees that

(a) every returned value is a proposed value,

(b) if only one value is proposed then this value must be committed,

[3] Formally, let $val(U)$ denote the set of non-$\perp$ values in a vector U. In a colorless task, for all input vectors I and I' and all output vectors O and O', such that $(I, O) \in \Delta$, $val(I) \subseteq val(I')$, $val(O') \subseteq val(O)$, we have $(I', O) \in \Delta$ and $(I, O') \in \Delta$.

(c) if a process commits on a value v, then every process that returns adopts v or commits v, and

(d) every correct process returns.

The CA abstraction can be implemented wait-free [12]. Moreover, CA can be viewed as a way to establish *safety* in shared-memory computations.

For example, consider a protocol where every processes goes through a series of instances of commit-adopt protocols, $CA_1, CA_2, \ldots$, one by one, where each instance receives a value adopted in the previous instance as an input (the initial input value for CA_1). One can easily see that once a value v is committed in some CA instance, no value other than v can ever be committed (properties (a) and (c) above). One the other hand, if at most one value is proposed to some CA instance, then this value must be committed by every process that takes enough steps (property (b) above).

This algorithm can be viewed as a *safe* version of consensus: every committed value is a proposed value and no two processes commit on different values (properties (a), (b) and (c) above). Given that every correct process goes from one CA instance to the other as long as it does not commit (property (d) above), we can boost the liveness guarantees of this protocol using external oracles.

In fact, the algorithm *per se* guarantees termination in every *obstruction-free* execution, i.e., assuming that eventually at most one process is taking steps. Moreover, we can build a consensus algorithm that terminates *almost always* if we allow processes to toss coins when choosing an input value for the next CA instance [2]. Also, if we allow a process to access an *oracle* (e.g., the Ω failure detector of [6]) that eventually elects a correct leader process, we get a live consensus algorithm.

2.4 The BG-simulation technique.

Another important tool used in this paper is *BG-simulation* [4, 5]. BG-simulation is a technique by which $k+1$ processes $s_1, \ldots, s_{k+1}$, called *simulators*, can wait-free simulate a *k-resilient* ($\mathcal{A}_{k\text{-}res}$-compliant) execution of any protocol *Alg* on m processes $p_1, \ldots, p_m$ ($m > k$). The simulation guarantees that each simulated step of every process p_j is either agreed upon by all simulators, or one less simulator participates further in the simulation for each step which is not agreed on.

The central building block of the simulation is the *BG-agreement* protocol. BG-agreement reminds consensus: processes propose values and agree one of the proposed values at the end. Indeed, the BG-agreement protocol ensures safety of consensus—every decided value was previously proposed, and no two different values are decided— but not liveness. If one of the simulators slows down while

executing BG-agreement, the protocol's execution at other correct simulators may "block" until the slow simulator finishes the protocol. If the slow simulator is faulty, no other simulator is guaranteed to decide.

Suppose the simulation tries to promote $m > k$ simulated processes in a fair (e.g., round-robin) way. As long there is a live simulator, at least $m - k$ simulated processes performs infinitely many steps of *Alg* in the simulated execution.

Recently the technique of BG-simulation was extended to show that any colorless task that can be solved assuming the $(k-1)$-resilient adversary can also be solved using read-write registers and k-set consensus objects [13].

3 Non-uniform failures in shared-memory systems

In this section, we overview several approaches to model non-uniform failures: dependent failure model of Junqueira and Marzullo [25], adversaries of Delporte et alii [9], and asymmetric progress conditions by Imbs et alii [24] and Taubenfeld [31].

3.1 Survivor Sets and Cores

Junqueira and Marzullo [26, 25] proposed to model non-uniform failures using the language of *survivor sets* and *cores*. A survivor set $S \subseteq \Pi$ if a set of processes such that:

(a) in some execution, S is the set of correct processes, and

(b) S is minimal: for every proper subset S' of S, there is no execution in which S' is the set of correct processes.

A collection $\mathcal{S}$ of survivor sets describes a system such that the set of correct processes in every execution contains a set in $\mathcal{S}$.

Respectively, a *core* C is a set of processes such that:

(a) in every execution, some process in C is correct, and

(b) C is minimal: for every proper subset C' of C, there is an execution in which every process in C' fails.

Thus, a core is a minimal set of processes that cannot be all faulty in any execution of our system. Note that the set of cores is unambiguously determined by the set of survivor sets.

A core is actually a *minimal hitting set* of the set system built of survivor sets, and a core of smallest size is a corresponding minimum hitting set. Determining minimum hitting set of a set system is known to be NP-complete [27].

The language of cores [26, 25] proved to be convenient in understanding the ability of a system with non-uniform failures to solve consensus or build a fault-tolerant replicated storage.

3.2 Adversaries

A more general way to model non-uniform failures was proposed by Delporte et al. [9]. Formally, an *adversary* defined for a set of processes Π is a non-empty set of process subsets $\mathcal{A} \subseteq 2^{\Pi}$. We say that an execution is $\mathcal{A}$-*compliant* if the *correct set*, i.e., the set of correct processes, in that execution belongs to $\mathcal{A}$. Thus, assuming an adversary $\mathcal{A}$, we only consider the set of $\mathcal{A}$-*compliant* executions. [4] By convention, we assume that in every execution, at least one process is correct, i.e., no adversary contains $\emptyset$.

Given a task T and an adversary $\mathcal{A}$, we say that T is $\mathcal{A}$-*resiliently solvable* if there is a protocol such that in every execution, the outputs match the inputs with respect to the specification of T, and in every $\mathcal{A}$-compliant execution, each correct process eventually produces an output.

It is easy to see that the language of survivor sets of [25] describes a special class of *superset-closed* adversaries. Formally, the set $\mathcal{SC}$ of superset-closed adversaries consists of all $\mathcal{A}$ such that for all $S \in \mathcal{A}$ and $S \subseteq S' \subseteq \Pi$, we have $S' \in \mathcal{A}$.

For example, consider the t-resilient adversary $\mathcal{A}_{t\text{-}res} = \{S \subseteq \Pi, |S| \geq n - t\}$. By definition, $\mathcal{A}_{t\text{-}res} \in \mathcal{SC}$. The survivor sets of $\mathcal{A}_{t\text{-}res}$ are all sets of $n-t$ processes, and the cores are all sets of $t+1$ processes. The $(n-1)$-resilient adversary $\mathcal{A}_{WF} = \mathcal{A}_{n-1\text{-}res}$ is also called *wait-free*. An $\mathcal{A}_{WF}$-resilient task solution must ensure that every process obtains an output in a finite number of its own steps, regardless of the behavior of the rest of the system.

Another example $\mathcal{A}_{L_p} = \{S \subseteq \Pi | p \in S\} \in \mathcal{SC}$ describing a system in which p never fails. $\mathcal{A}_{L_p}$ has one survivor set $\{p\}$ and one core $\{p\}$. Intuitively, p may then act as a correct leader in a consensus protocol. Thus, every task can be solved in the presence of $\mathcal{A}_{L_p}$ [18].

The *k-obstruction-free* adversary $\mathcal{A}_{k\text{-}OF}$ is defined as $\{S \subseteq \Pi \mid 1 \leq |S| \leq k\}$. In particular, $\mathcal{A}_{OF} = \mathcal{A}_{1\text{-}OF}$ allows for solving consensus [10]. Clearly, $\mathcal{A}_{k\text{-}OF}$ for $1 \leq k < n$ is not in $\mathcal{SC}$.

The "bimodal" adversary $\{pqr, p, q, r\}$ (Figure 1) is not in $\mathcal{SC}$ either: it contains the singleton p but not its supersets pq and pr.

[4] Note that in the original definition [9], an adversary is defined as a collection of *faulty sets*, i.e., the sets of processes that can fail in an execution. For convenience, we chose here an equivalent definition based on *correct sets*.

3.3 Failure patterns and environments

An adversary is in fact a special case of a *failure environment* introduced by Chandra et alii [6]. An environment $\mathcal{E}$ is a set of *failure patterns*. For a given run, a failure pattern F is a map that associates each time value $t \in \mathbb{T}$ with a set of processes crashed by time t. The set of correct processes, denoted $correct(F)$ is thus defined as $\Pi - \cup_{t\in\mathbb{T}} F(t)$.

Since an adversary $\mathcal{A}$ only defines sets of correct processes and does not specify the timing of failures, it can be viewed as a specific environment $\mathcal{E}_\mathcal{A}$ that is closed under changing the timing of failures. More precisely, $\mathcal{E}_\mathcal{A} = \{F \mid correct(F) \in \mathcal{A}\}$. Clearly, if $F \in \mathcal{E}_\mathcal{A}$ and $correct(F) = correct(F')$, then $F' \in \mathcal{E}_\mathcal{A}$.

Thus, we can rephrase the statement "task T can be solved $\mathcal{A}$-resiliently" as "task T can be solved in environment $\mathcal{E}_\mathcal{A}$". It is shown in [15] that, with respect to colorless tasks, all environments can be split into n equivalence classes, and each class j agrees on the set of tasks it can solve: namely, tasks that can be solved $(j-1)$-resiliently and not j-resiliently. Therefore, by applying [15], we conclude that each adversary belongs to one of such equivalence class. However, this characterization does not give us an explicit algorithm to compute the class to which a given adversary belongs.

3.4 Asymmetric progress conditions

Imbs et alii [24] introduced *asymmetric progress conditions* that allow us to specify different progress guarantees for different processes. Informally, for sets of processes X and Y, $X \subseteq Y \subseteq \Pi$, (X, Y)-liveness guarantees that every process in X makes progress regardless of other processes (wait-freedom for processes in X) and every process in $Y - X$ makes progress if it is eventually the only process in $Y - X$ taking steps (obstruction-freedom for processes in $Y - X$).

With respect to solving colorless tasks, it is easy to represent (X, Y)-liveness using the formalism of adversaries. The equivalent adversary $\mathcal{A}_{X,Y}$ consists of all subsets of Π that intersect with X and all sets $\{p_i\} \cup S$ such that $p_i \in Y - X$ and $S \subseteq \Pi - Y$. It is easy to see that a colorless task is (read-write) solvable assuming (X, Y)-liveness if and only if it is solvable in the presence of $\mathcal{A}_{X,Y}$.

Taubenfeld [31] introduced a refined condition that associates each process p_i with a set $\mathcal{P}_i$ of process subsets (each containing p_i). Then p_i is expected to make progress (e.g., output a value in a task solution) only if the current set of correct processes is in $\mathcal{P}_i$. Similarly, with respect to the question of solvability of colorless tasks, every such progress condition can be modeled as an adversary, defined simply as the union $\cup_i \mathcal{P}_i$.

4 Characterizing superset-closed adversaries

Intuitively, the size of a smallest-cardinality core of an adversary $\mathcal{A}$, denoted $csize(\mathcal{A})$, is related to its ability to "confuse" the processes (preventing them from agreement). Indeed, since in every execution, at least one process in a minimal core C is correct, we can treat C as a collection of leaders. But for a superset-closed adversary, every non-empty subset of C can be *the* set of correct processes in C in some execution. Therefore, intuitively, the system behaves like a wait-free system on $c = |C|$ processes, where c quantifies the "degree of disagreement" that we can observe among all the processes in the system.

In this section, we show that $csize(\mathcal{A})$ precisely captures the power of $\mathcal{A}$ with respect to colorless tasks. We overview two approaches to address this question, each interesting in its own right: using combinatorial topology and using shared-memory simulations.

4.1 A topological approach

Herlihy and Rajsbaum [20] derived a characterization of superset-closed adversaries using the Nerve Theorem of modern combinatorial topology [3]. A set of finite executions is modeled as a *simplicial complex*, a geometric (or combinatorial) structure where each simplex models a set of local states (*views*) of the processes resulting after some execution. This allows for reasoning about the power of a model using topological properties (e.g., connectivity) of simplicial complexes it generates.[5]

The model of [20] is based on *iterated* computations: each process p_i proceeds in (asynchronous) rounds, where every round r is associated with a shared array of registers $M[r,1],\ldots,M[r,n]$. When p_i reaches round r, it updates $M[r,i]$ with its current view and takes an atomic snapshot of $M[r,.]$. In the presence of a superset-closed adversary $\mathcal{A}$, the set of processes appearing in a snapshot should be an element of $\mathcal{A}$. We call the resulting set of executions the *$\mathcal{A}$-compliant iterated model*.

Naturally, given an adversary $\mathcal{A}$, it is easy to implement an iterated model with desired properties in the classical (non-iterated) shared memory model. To implement a round of the iterated model, every process writes its value in the memory and takes atomic snapshots until all processes in some survivor set (minimal element in $\mathcal{A}$) are observed to have written their values. The result of this snapshot is then returned. In an $\mathcal{A}$-compliant execution, this allows for simulating infinitely many iterated rounds.

[5] For more information on the applications of algebraic and combinatorial topology in distributed computing, check Maurice Herlihy's lectures at Technion [19].

Surprisingly, we can also use the $\mathcal{A}$-compliant iterated model to simulate an $\mathcal{A}$-compliant execution in the read-write model where *some* participating set of processes in $\mathcal{A}$ takes infinitely many steps (please check the wonderful simulation algorithm proposed recently by Gafni and Rajsbaum [17]). In particular, for the wait-free adversary $\mathcal{A}_{WF}$, the simulation is *non-blocking*: at least one participating process accepts infinitely many steps in the simulated execution.

Note that if the simulated $\mathcal{A}$-compliant execution is used for an $\mathcal{A}$-resilient protocol solving a given task, then we are guaranteed that at least one process obtains an output. But to solve a colorless task it is sufficient to produce an output for one participating process (all other participants may adopt this output). Thus:

Theorem 1. *[17] hosted Let $\mathcal{A}$ be a superset-closed adversary. A colorless task can be solved in the $\mathcal{A}$-compliant iterated model if and only if it can be solved in the $\mathcal{A}$-compliant model.*

This result allows us to apply the topological formalism as follows. The set of r-round executions of the $\mathcal{A}$-compliant iterated model applied to an initial simplex σ generates a *protocol complex* $\mathcal{K}_r(\sigma)$. By a careful reduction to the Nerve Theorem [3], $\mathcal{K}_r(\sigma)$ can be shown to be $(c-2)$-*connected*, i.e., $\mathcal{K}_r(\sigma)$ contains no "holes" in dimensions $c-2$ or less (any $(c-2)$-dimensional sphere can be continuously contracted to a point). The Nerve theorem establishes the connectivity of a complex from the connectivity of its components.

Roughly, the argument of [20] is built by induction on n, the number of processes. For a given adversary $\mathcal{A}$ on n processes with the minimal core size c, the $\mathcal{A}$-compliant protocol complex $\mathcal{K}_r(\sigma)$ can be represented as a union of protocol complexes, each corresponding to a sub-adversary of $\mathcal{A}$ on $n-1$ processes with core size $c-1$. By induction, each of these sub-adversaries is at least $(c-3)$-connected. Applying the Nerve theorem, we derive that $\mathcal{K}_r(\sigma)$ is $(c-2)$-connected. The base case $n=1$ and $c=1$ is trivial, since every non-empty complex is, by definition, (-1)-connected.

Thus, $\mathcal{K}_r(\sigma)$ is $(c-2)$-connected. Hence, no task that cannot be solved $(c-1)$-resiliently, in particular $(c-1)$-set consensus, allows for an $\mathcal{A}$-resilient solution [22].

Using the characterization of [22], we can reduce the question of $\mathcal{A}$-resilient solvability of a colorless task $T=(\mathcal{I},\mathcal{O},\Delta)$ to the existence of a continuous map f from $|skel^{c-1}(\mathcal{I})|$, the Euclidean embedding of the $(c-1)$-*skeleton* (the complex of all simplexes of dimension $c-1$ and less) of the input complex $\mathcal{I}$, to $|\mathcal{O}|$, the Euclidean embedding of the output complex $\mathcal{O}$, such that f is *carried by* Δ, i.e., $f(\sigma)\subseteq\Delta(\sigma)$. Indeed, the fact that of $\mathcal{K}_r(\sigma)$ is $(c-2)$-connected (and thus d-connected for all $0\le d\le c-2$) implies that every continuous map from d-sphere of $\mathcal{K}_r(\sigma)$ extends to the $(d+1)$-disk, for $0\le d\le c-2$. Intuitively, we can thus

inductively construct a continuous map from $|skel^{c-1}(\mathcal{I})|$ to $|\mathcal{O}|$, starting from any map sending a vertex of $\mathcal{I}$ to a vertex of $\mathcal{O}$ (for $d = 0$).

On the other hand, it is straightforward to construct an $\mathcal{A}$-resilient protocol solving a colorless task T, given a continuous map from the $(c-1)$-skeleton of the input complex of T to the output complex of T. Thus:

Theorem 2. *[20] An adversary $\mathcal{A} \in \mathcal{SC}$ with the minimal core size c allows for solving a colorless task $T = (\mathcal{I}, \mathcal{O}, \Delta)$ if and only if there is a continuous map from $|skel^{c-1}(\mathcal{I})|$ to $|\mathcal{O}|$ carried by Δ.*

Therefore, two adversaries in $\mathcal{A}, \mathcal{B} \in \mathcal{SC}$ with the same minimal core size c agree on the set of tasks they allow for solving, which is exactly the set of tasks that can be solved $(c-1)$-resiliently (since $csize(\mathcal{A}_{(c-1)\text{-}res}) = c$).

4.2 A simulation-based approach

It is comparatively straightforward to characterize superset-closed adversaries using classical BG-simulation [4, 5], and we present a complete proof below.

Theorem 3. *[14] Let $\mathcal{A}$ be a superset-closed adversary. A colorless task T is $\mathcal{A}$-resiliently solvable if and only if T is $(c-1)$-resiliently solvable, where c is the minimal core size of $\mathcal{A}$.*

Proof. Let a colorless task T be $(c-1)$-resiliently solvable, and let P_c be the corresponding algorithm. Let $C = \{q_1, \ldots, q_c\}$ be a minimal-cardinality core of $\mathcal{A}$ ($|C| = c$).

Let the processes in C BG-simulate the algorithm P_c running on all processes in Π. Here each simulator q_i tries to use its input value of task T as an input value of every simulated process [4, 5]. Since C is a core of $\mathcal{A}$, in every $\mathcal{A}$-compliant execution, at most $c-1$ simulators may fail. Since a faulty simulator results in at most one faulty simulated process, the produced simulated execution is $(c-1)$-resilient. Since P_c gives a $(c-1)$-resilient solution of T, at least one simulated process must eventually decide in the simulated execution. The output value is then adopted by every correct process. Moreover, the decided value is based on the "real" inputs of some processes. Since T is colorless, the decided values are correct with respect to the input values and, thus, we obtain an $\mathcal{A}$-resilient protocol to solve T.

For the other direction, suppose, by contradiction that there exists an $\mathcal{A}$-resilient protocol $P_{\mathcal{A}}$ to solve a colorless task T, but T is not possible to solve $(c-1)$-resiliently.

We claim that $\mathcal{A}_{(c-1)\text{-}res} \subseteq \mathcal{A}$, i.e., each $(c-1)$-resilient execution is $\mathcal{A}$-compliant. Suppose otherwise, i.e., some set S of $n-c+1$ processes is not in

$\mathcal{A}$. Since $\mathcal{A}$ is superset-closed, no subset of S is in $\mathcal{A}$ (otherwise, S would be in $\mathcal{A}$). No process in S belongs to any set in $\mathcal{A}$, thus, the smallest core of $\mathcal{A}$ must be a subset of $\Pi - S$. But $|\Pi - S| = c - 1$—a contradiction with the assumption that the size of a minimal cardinality core of $\mathcal{A}$ is c.

Thus, every $(c-1)$-resilient execution is also $\mathcal{A}$-compliant, which implies that $P_{\mathcal{A}}$ is in fact a $(c-1)$-resilient solution to T—a contradiction with the assumption that T is not $(c-1)$-resiliently solvable. □

Theorem 3 implies that adversaries in $\mathcal{SC}$ can be categorized into n equivalence classes, $\mathcal{SC}_1, \ldots, \mathcal{SC}_n$, where class $\mathcal{SC}_k$ corresponds to cores of size k. Two adversaries that belong to the same class $\mathcal{SC}_k$ agree on the set of colorless tasks they are able to solve, and it is exactly the set of all colorless task that can be solved $(k-1)$-resiliently.

5 Measuring the Power of Generic Adversaries

Let us come back to the "bimodal" adversary $\mathcal{A}_{BM} = \{pqr, p, q, r\}$ (Figure 1). Its only core is $\{p, q, r\}$. Does it mean that $\mathcal{A}_{BM}$ only allows for solving trivial (wait-free solvable) tasks? Not really: by splitting $\mathcal{A}_{BM}$ in two sub-adversaries $\mathcal{A}_{FF} = \{pqr\}$ and $\mathcal{A}_{OF} = \{p, q, r\}$ and running two consensus algorithms in parallel, one assuming no failures ($\mathcal{A}_{FF}$) and one assuming that exactly one process is correct ($\mathcal{A}_{OF}$), gives us a solution to 2-set consensus.

5.1 Solving consensus with $\mathcal{A}_{BM}$

But can we solve more in the presence of $\mathcal{A}_{BM}$? E.g., is there a protocol *Alg* that solves consensus $\mathcal{A}_{BM}$-resiliently? We derive that the answer is no by showing how processes, s_0 and s_1, can wait-free solve consensus through simulating an $\mathcal{A}_{BM}$-compliant execution of *Alg*. Initially, the two processes act as BG simulators [4, 5] trying to simulate an execution of *Alg* on *all* three processes p, q, and r. When a simulator s_i $(i = 0, 1)$ finds out that the simulation of some step is blocked (which means that the other simulator s_{1-i} started but has not yet completed the corresponding instance of BG-agreement), s_i switches to simulating a *solo execution* of the next process (in the round-robin order) in $\{p, q, r\}$. If the blocked simulation eventually resolves (s_{1-i} finally completes the instance of BG-agreement), then s_i switches back to simulating all p, q and r.

If no simulator blocks a simulated step forever, the simulated execution contains infinitely many steps of every process, i.e., the set of correct processes in it is $\{p, q, r\}$. Otherwise, eventually some simulated process forever runs in isolation and the set of correct processes in the simulated execution is $\{p\}$, $\{q\}$, or $\{r\}$. In

both cases, the simulated execution of *Alg* is $\mathcal{A}_{BM}$-compliant, and the algorithm must output a value, contradicting [11, 28]. This argument can be easily extended to show that $\mathcal{A}_{BM}$ cannot allow for solving any colorless task that cannot be solved 1-resiliently.

5.2 Disagreement power of an adversary

Thus, we need a more sophisticated criterion to evaluate the power of a generic adversary $\mathcal{A}$. Delporte et alii [9] proposed to evaluate the "disorienting strength" of an adversary $\mathcal{A}$ via its *disagreement power*.

Definition 1. *[9] The disagreement power of an adversary $\mathcal{A}$ is the largest k such that k-set consensus cannot be solved in the presence of $\mathcal{A}$.*

It is shown in [9] that adversaries of the same disagreement power agree on the sets of colorless task they allow for solving. The result is derived via a three-stage simulation. First, it is shown how an adversary can simulate any *dominating* adversary, where the domination is defined through an involved recursive inclusion property. Second, it is shown that every adversary $\mathcal{A}$ that does not dominate the k-resilient adversary[6] is strong enough to implement the anti-Ω_k failure detector that, in turn, can be used to solve k-set consensus [34]. Finally, it is shown that vector-Ω_k (a failure detector equivalent to anti-Ω_k) can be used to solve any colorless task that can be solved k-resiliently. Thus, the largest k such that k-set consensus cannot be solved $\mathcal{A}$-resiliently indeed captures the power of $\mathcal{A}$.

The characterization of adversaries proposed in [9] does not give a direct way of computing the disagreement power of an adversary $\mathcal{A}$ and it does not provide a direct $\mathcal{A}$-resilient algorithm to solve a colorless task T, when T is $\mathcal{A}$-resiliently solvable.

In the rest of this section, we give a simple algorithm to compute the disagreement power of an adversary. For convenience, we introduce notion of *set consensus power*, i.e., the smallest k such that k-set consensus can be solved in the presence of $\mathcal{A}$. Clearly, the disagreement power of $\mathcal{A}$ is the set consensus power of $\mathcal{A}$ minus 1.

5.3 Defining *setcon*

Let $\mathcal{A}$ be an adversary and let $S \subseteq P$ be any subset of processes. Then $\mathcal{A}_S$ denotes the adversary that consists of all elements of $\mathcal{A}$ that are subsets of S (including S itself if $S \in \mathcal{A}$). E.g., for $\mathcal{A} = \{pq, qr, q, r\}$ and $S = qr$, $\mathcal{A}_S = \{qr, q, r\}$. For

[6] Recall that the k-resilient adversary consists of all subsets of Π of size at least $n - k$.

$S \in \mathcal{A}$ and $a \in S$, let $A_{S,a}$ denote the adversary that consists of all elements of $\mathcal{A}_S$ that *do not* include a. E.g., for $\mathcal{A} = \{pq, qr, q, r\}$, $S = qr$, and $a = q$, $\mathcal{A}_{S,a} = \{r\}$.

Now we define a quantity denoted *setcon*($\mathcal{A}$), which we will show to be the set consensus power of $\mathcal{A}$. Intuitively, our goal is to split $\mathcal{A}$ into the minimal number k of sub-adversaries, such that every sub-adversary allows for solving consensus. Then $\mathcal{A}$ allows for solving k-set consensus, but not $(k-1)$-set consensus (otherwise, k would not be minimal).

Definition 2. *setcon*($\mathcal{A}$) *is defined as follows:*

- *If* $\mathcal{A} = \emptyset$, *then* $setcon(\mathcal{A}) = 0$
- *Otherwise,* $setcon(\mathcal{A}) = \max_{S \in \mathcal{A}} \min_{a \in S} setcon(\mathcal{A}_{S,a}) + 1$

Thus, *setcon*($\mathcal{A}$), for a non-empty adversary $\mathcal{A}$, is determined as $setcon(\mathcal{A}_{\overline{S},\overline{a}})+1$ where $\overline{S}$ is an element of $\mathcal{A}$ and $\overline{a}$ is a process in $\overline{S}$ that "max-minimize" $setcon(\mathcal{A}_{S,a})$. Note that for $\mathcal{A} \neq \emptyset$, $setcon(\mathcal{A}) \geq 1$.

We say that $S \in \mathcal{A}$ is *proper* if it is not a subset of any other element in $\mathcal{A}$. Let *proper*($\mathcal{A}$) denote the set of proper elements in $\mathcal{A}$. Note that since for all $S' \subset S$, $\min_{a \in S'} setcon(\mathcal{A}_{S',a}) \leq \min_{a \in S} setcon(\mathcal{A}_{S,a})$, we can replace $S \in \mathcal{A}$ with $S \in proper(\mathcal{A})$ in Definition 2.

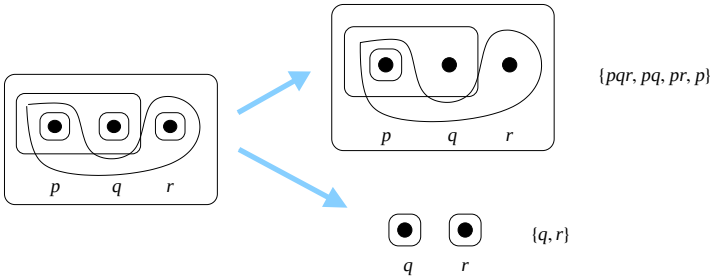

Figure 2: Adversary $\mathcal{A} = \{pqr, pq, pr, p, q, r\}$ decomposed in two sub-adversaries, $\{pqr, pq, pr, p\}$ and $\{q, r\}$, each with *setcon* = 1.

5.4 Calculating *setcon*($\mathcal{A}$): examples

Consider an adversary $\mathcal{A} = \{pqr, pq, pr, p, q, r\}$. It is easy to see that $setcon(\mathcal{A}) = 2$: for $S = pqr$ and $a = p$, we have $\mathcal{A}_{S,p} = \{q, r\}$ and $setcon(\mathcal{A}_{S,a}) = 1$. Thus, we decompose $\mathcal{A}$ into two sub-adversaries $\{pqr, pq, pr, p\}$ and $\{q, r\}$, each strong enough to solve consensus (Figure 2). Intuitively, in an execution where the correct set belongs to $\mathcal{A} - \mathcal{A}_{S,a} = \{pqr, pq, pr, p\}$, process p can act as a leader

for solving consensus. If the correct set belongs to $\mathcal{A}_{S,a} = \{q, r\}$ (either q or r eventually runs solo) then q and r can solve consensus using an obstruction-free algorithm. Running the two algorithms in parallel, we obtain a solution to 2-set consensus. The reader can easily verify that any other choice of $a \in pqr$ results in three levels of decomposition.

As another example, consider the t-resilient adversary $\mathcal{A}_{t\text{-}res} = \{S \subseteq \Pi, |S| \geq n - t\}$. It is easy to verify recursively that $setcon(\mathcal{A}_{t\text{-}res}) = t + 1$: at each level $1 \leq j \leq t + 1$ of recursion we consider a set S of $n - j + 1$ elements, pick up a process $p \in S$ and delegate the set of $n - j$ processes that do not include p to level $j+1$. At level $t+1$ we get one set of size $n-t$ and stop. Thus, $setcon(\mathcal{A}_{t\text{-}res}) = t+1$.

More generally, for any superset-closed adversary $\mathcal{A}$ ($\mathcal{A} \in \mathcal{SC}$), $setcon(\mathcal{A}) = csize(\mathcal{A})$, the size of a smallest-cardinality core of $\mathcal{A}$. To show this, we proceed by induction. The statement is trivially true for an empty adversary $\mathcal{A}$ with $csize(\mathcal{A}) = setcon(\mathcal{A}) = 0$. Now suppose that for all $0 \leq j < k$ and all $\mathcal{A}' \in \mathcal{SC}$ with $csize(\mathcal{A}') = j$, we have $setcon(\mathcal{A}') = j$. Consider $\mathcal{A} \in \mathcal{SC}$ such that $csize(\mathcal{A}) = k$. Note that the only proper element of $\mathcal{A}$ is the whole set of processes Π. Thus, $setcon(\mathcal{A}) = \min_{a \in \Pi} setcon(\mathcal{A}_{\Pi,a}) + 1$. By the induction hypothesis and the fact that $csize(\mathcal{A}) = k$, we have $\min_{a \in \Pi} setcon(\mathcal{A}_{\Pi,a}) = k - 1$. Thus, $setcon(\mathcal{A}) = k$.

Thus, by Theorem 3, *setcon*() indeed characterizes the disorienting power of adversaries $\mathcal{A} \in \mathcal{SC}$: a task is $\mathcal{A}$-resiliently solvable if and only if it is $(c - 1)$-resiliently solvable, where $c = setcon(\mathcal{A})$. In the rest of this section, we extend this result from $\mathcal{SC}$ to the universe of all adversaries.

5.5 Solving consensus with $setcon = 1$

Before we characterize the ability of adversaries to solve colorless tasks, we consider the special case of adversaries of $setcon = 1$.

Consider an adversary $\mathcal{A}$ and $S \in \mathcal{A}$. Suppose $csize(\mathcal{A}_S) = 1$, and let $\{a\}$ be a core of $\mathcal{A}_S$. Obviously, $\mathcal{A}_{S,a} = \emptyset$. On the other hand, if $\mathcal{A}_{S,a} = \emptyset$, then $\{a\}$ is a core of $\mathcal{A}_S$. Thus, $setcon(\mathcal{A}) = 1$ if and only if $\forall S \in \mathcal{A}$, $csize(\mathcal{A}_S) = 1$

Suppose $setcon(\mathcal{A}) = 1$. If S is the only proper element of $\mathcal{A}$, then we can easily solve consensus (and, thus, any other task [18]), by deciding on the value proposed by the only member of a core of $\mathcal{A}_S$. The process is guaranteed to be correct in every execution.

Now we extend this observation to the case when $\mathcal{A}$ contains multiple proper elements. The consensus algorithm, presented in Figure 3, is a "rotating coordinator" algorithm inspired by by Chandra and Toueg [7].

The algorithm proceeds in rounds. In each round r, every process p_i first tries to commit its current decision estimate in a new instance of commit-adopt CA_r. If p_i succeeds in committing the estimate, the committed value is written in the

Shared variables:
  D, initially $\perp$
  $R_1, \ldots, R_n$, initially $\perp$

propose(v)
1 $est := v$
2 $r := 0$
3 $S := P$
4 **repeat**
5 $r := r + 1$
6 $(flag, est) := CA_r.propose(est)$
7 **if** $flag = commit$ **then**
8 $D := est$; $return(est)$ {Return the committed value}
9 $R_i := (est, r)$
10 **wait until** $\exists S \in \mathcal{A}, \forall p_j \in S : R_j = (v_j, r_j)$ where $r_j \geq r$ **or** $D \neq \perp$
 {Wait until a set in $\mathcal{A}$ moves}
11 **if** $p_{r \mod n+1} \in S$ **then**
12 $est := v_{r \mod n+1}$ {Adopt the estimate of the current leader}
13 **until** $D \neq \perp$
14 $return(D)$

Figure 3: Consensus with a "one-level" adversary $\mathcal{A}$, $setcon(\mathcal{A}) = 1$

"decision" register D and returned. Otherwise, p_i adopts the returned value as its current estimate and writes it in R_i equipped with the current round number r. Then p_i takes snapshots of $\{R_1, \ldots, R_n\}$ until either a set $S \in \mathcal{A}$ reaches round r or a decision value is written in D (in which case the process returns the value found in D). If no decision is taken yet, then p_i checks if the coordinator of this round, $p_{r \mod n}$, is in S. If so, p_i adopts the value written in $R_{r \mod n}$ and proceeds to the next round.

The properties of commit-adopt imply that no two processes return different values. Indeed, the first round in which some process commits on some value v (line 8) "locks" the value for all subsequent rounds, and no other process can return a value different from v.

Suppose, by contradiction, that some correct process never returns in some $\mathcal{A}$-compliant execution e. Recall that $\mathcal{A}$-compliant means that some set in $\mathcal{A}$ is exactly the set of correct processes in e. If a process returns, then it has previously written the returned value in D. Since, in each round, a process performs a bounded number of steps, by our assumption, no process ever writes a value in D and every correct process goes through infinitely many rounds in e without returning.

Let $\bar{S} \in \mathcal{A}$ be the set of correct processes in e. After a round r' when all processes outside $\bar{S}$ have failed, every element of $\mathcal{A}$ evaluated by a correct process in line 10 is a subset of $\bar{S}$. Finally, since the minimal core size of $\mathcal{A}_{\bar{S}}$ is 1, all these elements of $\mathcal{A}$ overlap on some correct process p_j.

Consider round $r = mn + j \geq r' - 1$. In this round, p_j not only belongs to all sets evaluated by the correct processes, but it is also the coordinator ($j = r \mod n+1$). Thus, the only value that a process can propose to commit-adopt in round $r + 1$ is the value previously written by p_j in R_j. Hence, every process that returns from commit-adopt in round $r + 1$ must commit and return—a contradiction. Thus:

Theorem 4. *[14] If setcon$(\mathcal{A}) = 1$, then consensus can be solved $\mathcal{A}$-resiliently.*

5.6 Adversarial partitions

One way to interpret Definition 2 is to say that *setcon*$(\mathcal{A})$ captures the size of a minimal-cardinality partitioning of $\mathcal{A}$ into sub-adversaries $\mathcal{A}^1, \ldots, \mathcal{A}^k$, each of *setcon* $= 1$.

Indeed, for a proper set $S \in \mathcal{A}$, selecting an element $a \in S$ allows for splitting $\mathcal{A}_S$ into two sub-adversaries $\mathcal{A}_S - \mathcal{A}_{S,a}$ and $\mathcal{A}_{S,a}$. $\mathcal{A}_S - \mathcal{A}_{S,a}$ is the set of elements of $\mathcal{A}_S$ that contain a and, thus, *setcon*$(\mathcal{A}_S - \mathcal{A}_{S,a}) = 1$ (a can act as a leader). Moreover, selecting a so that *setcon*$(\mathcal{A}_{S,a})$ is minimized makes sure that $\mathcal{A}_{S,a} =$ *setcon*$(\mathcal{A}_S) - 1$.

Intuitively, $\mathcal{A}^1$, the first such sub-adversary, is the union of $\mathcal{A}_S - \mathcal{A}_{S,a}$, for all such proper $S \in \mathcal{A}$ and $a \in S$. Adversaries $\mathcal{A}_2, \ldots, \mathcal{A}_k$ are obtained by a recursive partitioning of all $\mathcal{A} - \mathcal{A}^1$. (A detailed description of this partitioning can be found in [14].)

Thus, given an adversary $\mathcal{A}$ such that $setcon(\mathcal{A}) = k$, we derive that $\mathcal{A}$ allows for solving k-set consensus. Just take the described above partitioning of $\mathcal{A}$ in to k sub-adversaries, $\mathcal{A}^1, \ldots, \mathcal{A}^k$ such that, for all $j = 1, \ldots, k$, $setcon(\mathcal{A}^j) = 1$. Then every process can run k parallel consensus algorithms, one for each $\mathcal{A}^j$, proposing its input value in each of these consensus instances (such algorithm exist by Theorem 4). Since the set of correct processes in every $\mathcal{A}$-compliant execution belongs to some $\mathcal{A}^j$, at least one consensus instance returns. The process decides on the first such returned value. Moreover, at most k different values are decided and each returned value was previously proposed. Thus:

Theorem 5. *[14] If $setcon(\mathcal{A}) = k$, then $\mathcal{A}$ allows for solving k-set consensus.*

5.7 Characterizing colorless tasks

But can we solve $(k-1)$-set consensus in the presence of $\mathcal{A}$ such that $setcon(\mathcal{A}) = k$? As shown in [14], the answer is no: $\mathcal{A}$ does not allow for solving any colorless task that cannot be solved $(k-1)$-resiliently. The result is derived by a simple application of BG simulation [4, 5].

The intuition here is the following. Suppose, by contradiction, that we are given an adversary $\mathcal{A}$ such that $setcon(\mathcal{A}) = k$ and a colorless task T that is solvable $\mathcal{A}$-resiliently but not $(k-1)$-resiliently. Let *Alg* be the corresponding $\mathcal{A}$-resilient algorithm. Then we can construct a $(k-1)$-resilient simulation of an $\mathcal{A}$-compliant execution of *Alg*. Roughly, we build upon BG-simulation, except that the *order* in which steps of *Alg* are simulated is not fixed in advance to be round-robin. Instead, the order is determined online, based on the currently observed set of participating processes.

We start with simulating steps of processes in $S \in \mathcal{A}$ such that $setcon(\mathcal{A}_S) = k$ (by Definition 2, such S exists). If the outcome of a simulated step of some process a cannot be resolved (the corresponding BG-agreement is blocked), we proceed to simulating processes in an element $S' \in \mathcal{A}_{S,a}$ with the largest *setcon* (if there is any). As soon as the blocked BG-agreement on the step of a resolves, the simulation returns to simulating S. Since $setcon(\mathcal{A}) = k$, we can obtain exactly k levels of simulation. Therefore, in a $(k-1)$-resilient execution, at most $k-1$ simulated processes (each in a distinct sub-adversary of $\mathcal{A}$) can be blocked forever. Since $\mathcal{A}$ allows for k such sub-adversaries, at least one set in $\mathcal{A}$ accepts infinitely many simulated steps. The resulting execution is thus $\mathcal{A}$-compliant, and

we obtain a $(k-1)$-resilient solution for T—a contradiction (detailed argument is given in [14]).

In fact, the set of colorless tasks that can be solved given an adversary $\mathcal{A}$ such that $setcon(\mathcal{A}) = k$ is *exactly* the set of colorless tasks that can be solved $(k-1)$-resiliently, but not k-resiliently. Indeed, $\mathcal{A}$ allows for solving k-set consensus, and we can employ the generic algorithm of [13] that solves any $(k-1)$-resilient colorless task using the k-set consensus algorithm as a black box. Thus:

Theorem 6. *[14] Let $\mathcal{A}$ be an adversary such that $setcon(\mathcal{A}) = k$ and T be a colorless task. Then $\mathcal{A}$ solves T if and only if T is $(k-1)$-resiliently solvable.*

Recall that the set consensus power of an adversary $\mathcal{A}$ is the smallest k such that $\mathcal{A}$ can solve k-set consensus. Theorem 6 implies:

Corollary 7. *The set consensus power of $\mathcal{A}$ is $setcon(\mathcal{A})$, and the disagreement power of $\mathcal{A}$ is $setcon(\mathcal{A}) - 1$.*

By Theorem 3, determining $setcon(\mathcal{A})$ may boil down to determining the minimum hitting set size of $\mathcal{A}$, and thus, by [27]:

Corollary 8. *Determining the set consensus power of an adversary is NP-complete.*

6 Concluding remarks

This survey primarily talks about colorless tasks (consensus, set agreement, simplex agreement, et cetera) in the read-write shared memory systems where processes may fail by crashing in a non-uniform (non-identical and correlated) way. We modeled such non-uniform failures using the language of adversaries [9] and we derived a complete characterization of an adversary via its set consensus power [14] (or, equivalently its disagreement power [9]).

The techniques discussed here can be extended to models where processes may also communicate through stronger objects than just read-write registers (e.g., k-process consensus objects). In particular, BG-simulation is used in [14] to capture the ability of leveled adversaries of [31] to prevent processes from solving consensus among n processes using k-process consensus objects ($k < n$).

Combinatorial topology proved to be a powerful instrument in analyzing a special class of superset-closed adversaries and colorless tasks, not only in read-write shared-memory models [20], but also in a variety of other models, including message-passing models and iterated models with k-set consensus objects.

However, the power of adversaries with respect to generic (not necessarily) colorless tasks is still poorly understood. Consider, for example, a task $\mathcal{T}_{pq}$ which requires processes p and q (in a system of three processes p, q, and r) to solve

consensus and allows r to output any value. The task is obviously not colorless: the output of r cannot always be adopted by p or q. The 2-obstruction-free adversary $\mathcal{A}_{2\text{-}OF} = \{pq, pr, qr, p, q, r\}$ does not allow for solving T_{pq}: otherwise, we would get a wait-free 2-process consensus algorithm. On the other hand, $\mathcal{A}_{pq} = \{pqr, pq, p, r\}$ (p is correct whenever q is correct) allows for solving T_{pq} (just use p as a leader for p and q). But $setcon(\mathcal{A}_{2\text{-}OF}) = setcon(\mathcal{A}_{pq}) = 2$!

One may say that the task T_{pq} is "asymmetric": it prioritizes outputs of some processes with respect to the others. Maybe our result would extend to symmetric tasks whose specifications are invariant under a permutation of process identifiers? Unfortunately, there are symmetric colored tasks that exhibit similar properties [33]. So we need a more fine-grained criterion than set consensus power to capture the power of adversaries with respect to colored tasks.

Finally, this paper focuses on non-uniform *crash* faults in asynchronous shared-memory systems. Non-uniform patterns of generic (Byzantine) types of faults are explored in the context of Byzantine quorum systems [29] (see also a survey in [32]) and secure multi-party computations [23]. Both approaches assume that a faulty process can deviate from its expected behavior in an arbitrary (Byzantine) manner. In particular, in [29], Malkhi and Reiter address the issues of non-uniform failures in the Byzantine environment by introducing the notion of a *fail-prone system* (*adversarial structure* in [23]): a set $\mathcal{B}$ of process subsets such that no element of $\mathcal{B}$ is contained in another, and in every execution some $B \in \mathcal{B}$ contains all faulty processes. Determining the set of tasks solvable in the presence of a given generic adversarial structure is an interesting open problem.

References

[1] Yehuda Afek, Hagit Attiya, Danny Dolev, Eli Gafni, Michael Merritt, and Nir Shavit. Atomic snapshots of shared memory. *Journal of the ACM*, 40(4):873–890, 1993.

[2] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC '83: Proceedings of the annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[3] A. Björner. *Topological methods*, pages 1819–1872. MIT Press, Cambridge, MA, USA, 1995.

[4] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*, pages 91–100. ACM Press, May 1993.

[5] Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Computing*, 14(3):127–146, 2001.

[6] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722, July 1996.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[8] Soma Chaudhuri. More *choices* allow more *faults*: Set consensus problems in totally asynchronous systems. *Information and Computation*, 105(1):132–158, 1993.

[9] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. *Distributed Computing*, 24(3-4):137–147, 2011.

[10] Faith Ellen Fich, Victor Luchangco, Mark Moir, and Nir Shavit. Obstruction-free algorithms can be practically wait-free. In *Proceedings of the International Symposium on Distributed Computing*, pages 493–494, 2005.

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

[12] Eli Gafni. Round-by-round fault detectors (extended abstract): Unifying synchrony and asynchrony. In *Proceedings of the 17th Symposium on Principles of Distributed Computing*, 1998.

[13] Eli Gafni and Rachid Guerraoui. Generalized universality. In *Proceedings of the 22nd international conference on Concurrency theory*, CONCUR'11, pages 17–27, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] Eli Gafni and Petr Kuznetsov. Turning adversaries into friends: Simplified, made constructive, and extended. In *OPODIS*, pages 380–394, 2010.

[15] Eli Gafni and Petr Kuznetsov. On set consensus numbers. *Distributed Computing*, 24(3-4):149–163, 2011.

[16] Eli Gafni and Petr Kuznetsov. Relating L-Resilience and Wait-Freedom via Hitting Sets. In *ICDCN*, pages 191–202, 2011.

[17] Eli Gafni and Sergio Rajsbaum. Distributed programming with tasks. In *OPODIS*, pages 205–218, 2010.

[18] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):123–149, January 1991.

[19] Maurice Herlihy. Advanced topics in distributed algorithms. Technion Lecture, 2011. http://video.technion.ac.il/Courses/Adv_Topics_in_Dist_Algorithms.html.

[20] Maurice Herlihy and Sergio Rajsbaum. The topology of shared-memory adversaries. In *PODC*, pages 105–113, 2010.

[21] Maurice Herlihy and Nir Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 111–120, May 1993.

[22] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *Journal of the ACM*, 46(2):858–923, 1999.

[23] Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *PODC*, pages 25–34, 1997.

[24] Damien Imbs, Michel Raynal, and Gadi Taubenfeld. On asymmetric progress conditions. In *PODC*, 2010.

[25] Flavio Junqueira and Keith Marzullo. A framework for the design of dependent-failure algorithms. *Concurrency and Computation: Practice and Experience*, 19(17):2255–2269, 2007.

[26] Flavio Paiva Junqueira and Keith Marzullo. Designing algorithms for dependent process failures. In *Future Directions in Distributed Computing*, pages 24–28, 2003.

[27] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[28] M.C. Loui and H.H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[29] Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[30] Michael Saks and Fotios Zaharoglou. Wait-free *k*-set agreement is impossible: The topology of public knowledge. In *Proceedings of the 25th ACM Symposium on Theory of Computing*, pages 101–110. ACM Press, May 1993.

[31] Gadi Taubenfeld. The computational structure of progress conditions. In *DISC*, 2010.

[32] Marko Vucolić. The origin of quorum systems. *Bulletin of EATCS*, 101:125–147, June 2010.

[33] Piotr Zieliński. Sub-consensus hierarchy is false (for symmetric, participation-aware tasks). https://sites.google.com/site/piotrzielinski/home/symmetric.pdf.

[34] Piotr Zieliński. Anti-*omega*: the weakest failure detector for set agreement. *Distributed Computing*, 22(5-6):335–348, 2010.