Distrib. Comput. (2011) 24:137–147
DOI 10.1007/s00446-010-0122-4

The disagreement power of an adversary

Carole Delporte-Gallet · Hugues Fauconnier · Rachid Guerraoui · Andreas Tielmann

Received: 19 November 2009 / Accepted: 11 November 2010 / Published online: 7 December 2010

Abstract At the heart of distributed computing lies the fundamental result that the level of agreement that can be obtained in an asynchronous shared memory model where t processes can crash is exactly $t+1$. In other words, an adversary that can crash any subset of size at most t can prevent the processes from agreeing on t values. But what about all the other $2^{2^n-1}-(n+1)$ adversaries that are not uniform in this sense and might crash certain combination of processes and not others? This paper presents a precise way to classify all adversaries. We introduce the notion of *disagreement power*: the biggest integer k for which the adversary can prevent processes from agreeing on k values. We show how to compute the disagreement power of an adversary and derive n *equivalence* classes of adversaries.

1 Introduction

The theory of distributed computing is largely related to determining what can be computed against a specific adversary. Most results so far have been devoted to *one* specific form of adversaries: those that can control any subset of size t of the processes, i.e., the *t-failures* adversary. In particular, a seminal result in distributed computing says that the level of agreement that can be obtained in a shared memory model where t processes can crash is exactly $t+1$ [3,12,15]. In other words, an adversary that can crash any subset of size at most t can prevent the processes from agreeing on t values. In the case of consensus for instance ($t=1$), this translates into the FLP impossibility result [9].

In a sense, these results are very incomplete. Indeed, the t-failures assumption covers only the n "uniform" adversaries in a system of size n. What about the other $2^{2^n-1}-(n+1)$ adversaries that can crash certain subsets of processes of a certain size but not others of the same size? In particular, given any adversary $\mathscr{A}$, for what k does $\mathscr{A}$ prevent k-set agreement [5]? Beyond intellectual curiosity, the study of adversaries that are "non-uniform" is practically motivated by modern multicore architectures where the failures of processes in the same core are correlated [2,10,13].

This paper characterizes the power of an adversary $\mathscr{A}$ by the biggest k for which k-set agreement cannot be solved with $\mathscr{A}$, which we call here the *disagreement power* of $\mathscr{A}$. We show how to automatically compute the disagreement power of an adversary and group adversaries into n equivalence classes in a system of size n. Adversaries within the same class solve the same set of *colorless* tasks. Intuitively, in a colorless task [4,11], any process can adopt any other process' input or output value. Colorless tasks include k-set agreement and loop-agreement [11].

Determining the disagreement power of certain adversaries is trivial. For others, it is not. Consider, in a system of 3 processes, $\{p_1, p_2, p_3\}$, an adversary $\mathscr{A}$ that can fail no process, both processes p_2 and p_3, or process p_1, i.e., $\mathscr{A} = \{\emptyset, 23,$[1]$1\}$. It is easy to show that $\mathscr{A}$ can prevent consensus but not 2-set agreement. In this sense, adversary $\mathscr{A}$ has the same disagreement power as the 1-failure adversary, namely, 1. Consider now a more involved scenario: a system of 4 processes and an adversary $\mathscr{A}'$ that can fail any element of $\{\emptyset, 4, 23, 14, 12, 134, 124, 123\}$. What is the disagreement

C. Delporte-Gallet · H. Fauconnier · A. Tielmann (✉)
LIAFA, Université Paris Diderot, Paris, France
e-mail: tielmann@liafa.jussieu.fr

R. Guerraoui
School of Computer and Communication Sciences, EPFL, Lausanne, Switzerland

[1] When appropriate, we will use $ij\ldots$ as a shorthand for the set $\{p_i, p_j, \ldots\}$.

power of $\mathscr{A}'$? Using the results of this paper, it can be shown that it is also 1.

We give a general characterization of adversaries that enables us to automatically compute their disagreement power. Namely, we introduce a *structural predicate* on adversaries, parameterized by an integer k, and which, intuitively, checks that for any set of faulty processes of size less than or equal to k, there is some adequate *matching* set of the adversary. If there exists such matching sets, then the structure of the adversary is in some sense similar to the structure of the uniform k-failures adversary. We prove that any adversary that satisfies the predicate has disagreement power k.

We first show (sufficient condition) that if k-set agreement can be solved with some adversary that satisfies the predicate for k, then k-set agreement can be solved with the k-failures adversary which, in turn, is known to be impossible [3,12,15]. Hence, an adversary that satisfies the structural predicate has disagreement power at least k. For this, we use a new simulation between adversaries, which we call the *conservative back-off simulation*, and which we believe is interesting in its own right. The idea underlying our simulation is the following: a process backs-off and skips its simulation step if the process thinks that it is in some faulty-set of an adversary where the simulated algorithm is known to work. Conversely (necessary condition), we show how to solve k-set agreement with any adversary $\mathscr{A}$ that does not satisfy the predicate for k. We do this by showing how to implement failure detector k-anti-Ω [17], known to implement k-set agreement. (Each query to k-anti-Ω returns $n-k$ process ids; the specification ensures that there is a correct process whose id is eventually never output.)

We then use our characterization to split the set of all adversaries into n disjoint *equivalence* classes, one for every level of disagreement: we show that for any two adversaries with the same disagreement power, exactly the same set of (colorless) tasks can be solved. Intuitively, in a colorless task [4,11] any process can adopt any input or output value of any other process without violating the task specification. The key to our proof of the equivalence is that, for every adversary with disagreement power k, it is possible to simulate a wait-free system of $k+1$ processes. This can simulate every other k-failure adversary [4,6]. Technically, this is achieved by implementing $(k+1)$-anti-Ω for the adversary and translating it into a vector of $k+1\Omega$ failure detectors [7] of which at least one is a "real" Ω (i.e., it eventually outputs the same correct process everywhere). Then, each of the $k+1$ simulated processes can be associated with one of the Ω's and a consensus-object can be built to agree on the simulated steps of such a process.

Since we can compute automatically the disagreement power of an adversary (using our structural predicate), we can thus automatically derive results for an "exotic" adversary using known results about a more orthodox ("uniform") adversary with the same disagreement power.

Indirectly, our partitioning contributes to the idea that a very small subset of results and ad-hoc proofs in distributed computing should suffice to derive all others. In particular, if indeed needed to reason about set agreement for the "wait-free" adversary ($(n-1)$-set agreement), topology is not needed for all the other ones. Results concerning other k-failures ("uniform") adversaries can be deduced by [4,6], whereas results for all of the other ("non-uniform") $2^{2^n-1}-(n+1)$ adversaries can be deduced from our characterization.

The remainder of the paper is structured as follows. We first define our model in Sect. 2. We then introduce our notion of disagreement power and our structural predicate in Sect. 3. We present our conservative back-off simulation and use it in Sect. 4 to show that any adversary that satisfies the predicate for k can be reduced to the k-failure adversary (thus the predicate is sufficient for the simulation). We show in Sect. 5 how to implement k-set agreement with any adversary that does not satisfy the predicate (therefore, the predicate is necessary). We then show that adversaries with the same disagreement power are actually in the same equivalence class in Sect. 6 and conclude the paper with some general remarks in Sect. 7.

2 Model and definitions

We assume a system of deterministic processes that communicate asynchronously using read-write atomic registers. We recall below the necessary elements to describe our model and introduce the notion of an adversary.

Processes and registers. Our system consists of n processes $\Pi = \{p_1, p_2, \ldots, p_n\}$ sharing atomic registers. We assume that processes might crash, i.e., fail by prematurely halting. Processes that crash are called faulty and a process that never crashes is said to be correct.

Adversaries and runs. Intuitively, an *adversary* can choose which set of processes will crash. More precisely, we represent an adversary as a set of sets of processes (we call these sets *faulty-sets*) and the adversary can choose one of these faulty-sets. Here, we consider only adversaries $\mathscr{A}$ for which there is always at least one correct process, i.e., $\Pi \notin \mathscr{A}$.

A run of an algorithm A is an infinite sequence of steps of the processes. Given an adversary $\mathscr{A}$, a run is associated with a set of processes $a \in \mathscr{A}$ that will crash. This set is chosen by the adversary and the processes in a may crash at any time. The set of processes that take infinitely many steps in some run associated with a is then exactly $\Pi \backslash a$.

The classical n process k-failure adversary, denoted $\mathscr{U}_k^n$ is the adversary for which at most k ($0 \leq k \leq n-1$) processes may crash: $\mathscr{U}_k^n = \{u \subseteq \{p_1, \ldots, p_n\} \mid |u| \leq k\}$. Where the

number of processes is clear from the context, we will omit the n (i.e., $\mathscr{U}_k := \mathscr{U}_k^n$).

Tasks. A task is a tuple $(\mathfrak{I}, \mathfrak{O}, \Delta)$, where $\mathfrak{I}$ is a set of vectors of input values and $\mathfrak{O}$ is a set of vectors of output values such that the value of every process p_i corresponds to the i-th entry of a vector. Δ is a binary relation from $\mathfrak{I}$ to $\mathfrak{O}$. Then, a task is solved if for each input vector $I \in \mathfrak{I}$, an output vector $O \in \mathfrak{O}$ is computed such that $O \in \Delta(I)$. Generally, we say that algorithm A solves a task T against adversary $\mathscr{A}$ if every run of A associated with each $a \in \mathscr{A}$ satisfies the specification of T (we say also A implements T against adversary $\mathscr{A}$).

In the following, we restrict ourselves to colorless tasks (also called convergence tasks) [4,11]. Let $val(V)$ be the set of values in some vector V. A *colorless* task is such that if for all $I \in \mathfrak{I}$ and for all I' with $val(I') \subseteq val(I)$ then $I' \in \mathfrak{I}$ and $\Delta(I') \subseteq \Delta(I)$. Furthermore, if $O \in \Delta(I)$, then for every O' with $val(O') \subseteq val(O)$ we have $O' \in \mathfrak{O}$ and $O' \in \Delta(I)$. As a result, a colorless task can be specified independently of the number of processes. In this sense, such a task specifies a *family of tasks*, one for every possible number of processes.

k-set agreement. The canonical example of a colorless task is *k-set agreement*. Let S be a set of values with $|S| = k+1$. In k-set agreement, $\mathfrak{I}$ and $\mathfrak{O}$ are the sets of all vectors of values from S such that for all $O \in \mathfrak{O}$, $|val(O)| \leq k$ and for every $I \in \mathfrak{I}$ we have $O \in \Delta(I)$ iff $val(O) \subseteq val(I)$.

Consensus is 1-set agreement. k-set agreement can be solved against $\mathscr{U}_l$ iff $0 \leq l \leq k-1$ [3,12,15].

In our proofs, we will use a failure detector called k-anti-Ω [17]. It is a distributed oracle that gives the processes information about failures [8]. Each query to k-anti-Ω returns $n-k$ process ids, with the guarantee that there is a correct process whose id is returned only a finite number of times at correct processes. If $k = 1$, then k-anti-Ω is equivalent to the eventual leader failure detector named Ω, the weakest failure detector for consensus [7,14]. If $k = n-1$, k-anti-Ω is anti-Ω, the weakest failure detector to solve $(n-1)$-set agreement [17]. In general, k-anti-Ω is sufficient to solve k-set agreement [17].

3 Disagreement power

We define the *disagreement power* of an adversary $\mathscr{A}$ to be the maximal k $(0 \leq k < n)$ for which it is impossible to implement k-set agreement against $\mathscr{A}$. More precisely:

Definition 1 We say that an adversary $\mathscr{A}$ has *disagreement power* k $(0 \leq k < n)$, denoted $dis(\mathscr{A})$, if (1) it is impossible to implement k-set agreement against $\mathscr{A}$, and (2) it is possible to implement $(k+1)$-set agreement against $\mathscr{A}$.

If an adversary cannot prevent agreement for any k, then we say that its disagreement power is 0. As established in [3,12,15], it is possible to implement $(k+1)$-set agreement against $\mathscr{U}_k$ but it is impossible to implement k-set agreement against $\mathscr{U}_k$. Hence, the disagreement power of $\mathscr{U}_k$ is k.

Proposition 1 $dis(\mathscr{U}_k) = k$.

To compare the power of two adversaries, we define what it means for an adversary to be *stronger* than another adversary:

Definition 2 An adversary $\mathscr{A}$ is *stronger* than an adversary $\mathscr{B}$ (denoted $\mathscr{A} \succcurlyeq \mathscr{B}$) if every colorless task that can be solved against $\mathscr{A}$ can be solved against $\mathscr{B}$.

This relation is clearly transitive. We also compare our adversaries with a structural *domination* property without considering the tasks that they can solve. The interesting point, as we will show later, is that this property captures exactly the disagreement power of an adversary. For our domination property, we implicitly assume that both adversaries are built upon the same set of processes Π.

Definition 3 Let $\mathscr{A}$ and $\mathscr{B}$ be any two adversaries. We say that a faulty-set $a \in \mathscr{A}$ *dominates* a faulty-set $b \in \mathscr{B}$ in $\mathscr{A}$ and $\mathscr{B}$ (denoted $D(a, \mathscr{A}, b, \mathscr{B})$), if $a \supseteq b$ and

$\forall b' \in \mathscr{B}$ such that $b' \supsetneq b$, $\exists a' \in \mathscr{A}$ such that $a' \supseteq a$ and $D(a', \mathscr{A}, b', \mathscr{B})$.

In the base case, when there is no strict superset of b in $\mathscr{B}$, this translates to $a \supseteq b$. Where $\mathscr{A}$ and $\mathscr{B}$ are clear from the context, we will simply write $D(a, b)$. With a slight abuse of the D-symbol, we extend the notion of domination to adversaries:

Definition 4 We say that an adversary $\mathscr{A}$ *dominates* an adversary $\mathscr{B}$ (denoted $D(\mathscr{A}, \mathscr{B})$) if and only if the following property is satisfied:

$\forall b \in \mathscr{B}, \exists a \in \mathscr{A}$ such that $D(a, \mathscr{A}, b, \mathscr{B})$.

This property is intricate. First, we give an example in which the domination property holds for two adversaries.

Example 1 Assume $n = 4$ and consider the following adversaries (we use $ij\ldots$ as a shorthand for the set $\{p_i, p_j, \ldots\}$):

$\mathscr{A} = \{\emptyset, 12, 34, 123, 124, 134, 234\}$

$\mathscr{U}_2 = \{\emptyset, 1, 2, 3, 4, 12, 13, 14, 23, 24, 34\}$

In this example (also shown in Fig. 1), $D(\mathscr{A}, \mathscr{U}_2)$, i.e., for every $u \in \mathscr{U}_2$ there exists an $a \in \mathscr{A}$ such that $D(a, u)$ (e.g. $D(\emptyset, \emptyset)$, $D(12, 2)$ and $D(123, 13)$).

It does not follow that $D(\mathscr{A}, \mathscr{B})$ holds if, for all $b_0 \subsetneq b_1 \cdots \subsetneq b_x$ in $\mathscr{B}$ there exist $a_0 \subseteq a_1 \cdots \subseteq a_x$ in $\mathscr{A}$ such that $b_i \subseteq a_i$ for all i. To see this, consider the following example with $n = 3$:

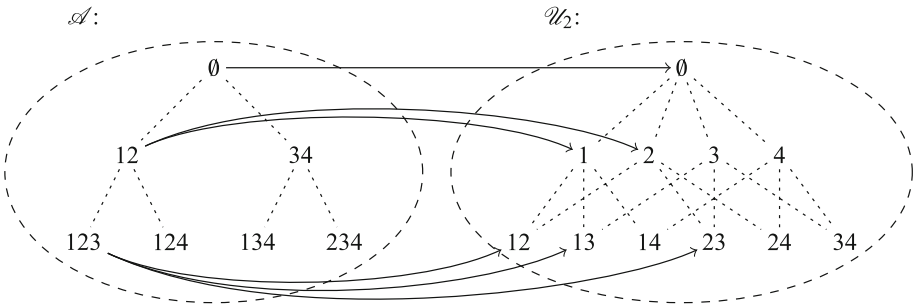

Fig. 1 Adversaries $\mathscr{A}$ and $\mathscr{U}_2$ from Example 1 with some exemplary domination relations

Example 2

$\mathscr{A} = \{\emptyset, 2, 12, 13, 23\}$
$\mathscr{U}_2 = \{\emptyset, 1, 2, 3, 12, 13, 23\}$

In this example, for all $u_0 \subsetneq u_1 \subsetneq u_2$ there exist $a_0 \subseteq a_1 \subseteq a_2$ and $u_i \subseteq a_i$. But $\neg D(\mathscr{A}, \mathscr{U}_2)$, because for all $a \in \mathscr{A}$ we have $\neg D(a, 3)$. Consider for example $a = 13$. We have $3 \subseteq 13$, but for 23, a superset of 3 in $\mathscr{U}_2$, there is no superset of 13 in $\mathscr{A}$ that contains 23.

In our proofs, we use two direct consequences of the domination relation. These are expressed by the following two lemmas:

Lemma 1 *Let $\mathscr{A}$ and $\mathscr{B}$ be any two adversaries. Then, for every pair $b, b' \in \mathscr{B}$ with $b \subsetneq b'$, and for every $a \in \mathscr{A}$ with $D(a, \mathscr{A}, b, \mathscr{B})$:*

$$\exists a' \in \mathscr{A}, \quad a' \supseteq a : D(a', \mathscr{A}, b', \mathscr{B}).$$

Proof This property follows immediately from the definition of the domination relation. □

Lemma 2 *Let $\mathscr{A}$ and $\mathscr{B}$ be any two adversaries. Then, for every $a \in \mathscr{A}$ and for every $b \in \mathscr{B}$ with $\neg D(a, \mathscr{A}, b, \mathscr{B})$:*

$a \supseteq b$ implies that there exists some $b' \in \mathscr{B}$ with $b' \supsetneq b$ such that $\forall a' \in \mathscr{A}$ with $a' \supseteq a$ it holds that $\neg D(a', \mathscr{A}, b', \mathscr{B})$.

Proof This property follows immediately from the definition of the domination relation. □

Interestingly, concerning adversary $\mathscr{U}_k$, our definitions induce the following property:

Theorem 1 *Consider any k with $1 \leq k \leq n-1$ and any element $u \in \mathscr{U}_k$. Then $\neg D(\mathscr{U}_k \backslash \{u\}, \mathscr{U}_k)$.*

Proof It is sufficient to show that we have for all $a \in \mathscr{U}_k \backslash \{u\}$: $\neg D(a, u)$. Assume by contradiction $D(a, u)$ for some $a \in \mathscr{U}_k \backslash \{u\}$. Then, by definition of $D(a, u)$: $a \supseteq u$. As $a \in \mathscr{U}_k \backslash \{u\}$, we have $a \supsetneq u$ and thus there exists a process $p \in a$ and $p \notin u$. Consider two cases:

Case 1: $|u| = k$: In this case, $\nexists a \in \mathscr{U}_k \backslash \{u\}$ such that $a \supseteq u$. A contradiction to the fact that $D(a, u)$.

Case 2: $|u| < k$: As $k \leq n-1$, we can construct $u' = u \cup x$ such that:

(i) $|u'| = k$ (and thus $u' \in \mathscr{U}_k$) and
(ii) $p \notin x$.

Thus, $|u' \cup \{p\}| = k+1$. Since $p \in a$, we have $a \cup u' \supseteq \{p\} \cup u'$ and therefore

$$|a \cup u'| > k. \tag{1}$$

By definition of $D(a, u)$ applied to u', there exists some $a' \in \mathscr{U}_k \backslash \{u\}$ with $a' \supseteq a$, $D(a', u')$ and thus $a' \supseteq u'$. Therefore, we have $a' \supseteq a \cup a' \supseteq a \cup u'$ and with (1) this implies that $|a'| > k$. A contradiction to the fact that $a' \in \mathscr{U}_k \backslash \{u\}$. □

4 The conservative back-off simulation (sufficient condition)

In this section, we show that if, for adversaries $\mathscr{A}$ and $\mathscr{B}$, we have $D(\mathscr{A}, \mathscr{B})$, then $\mathscr{A}$ is stronger than $\mathscr{B}$. Given that k-set agreement cannot be implemented against $\mathscr{U}_k$, we get a sufficient condition for the impossibility of implementing k-set agreement. Namely, if $D(\mathscr{A}, \mathscr{U}_k)$, then k-set agreeement cannot be implemented against $\mathscr{A}$.

Assume $D(\mathscr{A}, \mathscr{B})$ for some adversaries $\mathscr{A}$ and $\mathscr{B}$ over the same set of processes Π. Let A be any algorithm which solves a colorless task T against $\mathscr{A}$. The *conservative back-off simulation* described in Algorithm 1 solves T with A against $\mathscr{B}$.

The goal of the simulation is to identify, in every possible run with a set of faulty processes $b^* \in \mathscr{B}$, a set of processes $a^* \in \mathscr{A}$ with $b^* \subseteq a^*$. Hence, the processes outside a^* can use the given algorithm which is known to terminate for every $a^* \in \mathscr{A}$. The processes in a^* that are not in b^* can then just back-off and omit to take simulation steps, since the others are enough to ensure termination. Thus, termination is achieved by simply letting some correct processes take

only finitely many steps, i.e., to simulate their crashes. The challenge in this simulation is to find an appropriate set a^*. While it is easy to find sets of processes that contain only correct processes (e.g. by simply choosing the processes that recently took steps), it is in general impossible to eventually agree on such a set (otherwise, problems like consensus could be easily solved). To circumvent this problem, we take advantage of two things:

- It is not necessary to agree exactly on some a^*. It is sufficient if each process can find sequences $a_1 \subseteq a_2 \subseteq \ldots$ of sets such that eventually a^* is always contained in every such sequence. If some process overestimates a^*, i.e., it takes a step because it is not faulty in some set $a \supsetneq a^*$, this is no problem, because it is also not faulty in a^*. To keep the processes from underestimating a^* (i.e., assuming a set $a \subsetneq a^*$), we let them check between every pair of two simulation steps if all other processes not in a are still alive.
- The sets in $\mathscr{A}$ and $\mathscr{B}$ are in a certain relation, because $D(\mathscr{A}, \mathscr{B})$.

To determine a^*, we first narrow down the possibilities for b^* in the run. This is achieved by simply using step-counters (denoted STEPC). The current estimations are stored in the set POSSIBLY- FAULTY. Then, starting from the smallest set $b \in$ POSSIBLY- FAULTY, every process tries to stepwise approximate a^*.

In these steps, we need our property $D(\mathscr{A}, \mathscr{B})$. We maintain a set FAULTY that contains our estimates of a^*, i.e., the set of processes who shall consider themselves as faulty. For every set b that is in POSSIBLY- FAULTY, starting from the smallest, some $a \in \mathscr{A}$ with $D(a, b)$ that is a superset of every other previously added element in FAULTY is deterministically chosen (depending only on the state of FAULTY and POSSIBLY- FAULTY) and added to FAULTY. Since $D(a, b)$, and every possible next b' in POSSIBLY- FAULTY is a superset of b, it is guaranteed that there will always be an $a' \in \mathscr{A}$ that is a superset of a and $D(a', b')$ (compare Lemma 1). This sequence of a's is stored in FAULTY. Since the subsets of b^* in POSSIBLY- FAULTY are stable (i.e., they are eventually always in POSSIBLY- FAULTY), even if the supersets of b^* change infinitely often, the a added in the step where b^* is considered is such that $b^* \subseteq a$. Thus, the a^* we are trying to seek is just the smallest set in FAULTY where $b^* \subseteq a^*$. Although we do not know which one of the elements of FAULTY it is, it is safe for a process p to take a step if it does not belong to some $a \in$ FAULTY and has reason to believe that all other processes that are not in a are alive. This is simply achieved by determining which processes took steps since p's last simulation step using the variable ALIVE and the state LASTSIMSTEPS at the last simulation step. A process not in a^* will not block here forever, because all non-faulty processes increase their step-counters infinitely often.

If some process decides, it writes its decision value into a special register. If some other process observes that another process has decided, it adopts its decision value and decides also.

Consider the adversaries from Example 1: $\mathscr{A} = \{\emptyset, 12, 34, 123, 124, 134, 234\}$, $\mathscr{U}_2$ and $n = 4$. If the actual faulty-set u^* is 3, then eventually POSSIBLY- FAULTY can only be: $\{\emptyset, 3, 23\}$, $\{\emptyset, 3, 13\}$ or $\{\emptyset, 3, 34\}$, because process p_3 takes the least number of steps. Thus, $\emptyset$ and $u^* = 3$ are eventually always contained in POSSIBLY- FAULTY at all correct processes.

The only set from $\mathscr{A}$ that dominates $\emptyset \in \mathscr{U}_2$ is $\emptyset$. The only one that dominates 3 is 34. Thus, the set FAULTY will be $\{\emptyset, 34, 134\}$ or $\{\emptyset, 34, 234\}$ respectively. Therefore, a set $a^* = 34$ with $a^* \supseteq u^* = 3$ is eventually always contained in FAULTY at all correct processes.

For the three processes p_1, p_2 and p_4 that take infinitely many steps, eventually ALIVE $\subseteq$ 124, because process p_3 takes only finitely many steps. If processes p_1, p_2 or p_4 take only finitely many simulation steps, then we have eventually ALIVE $=$ 124 at these processes (because the step-counters keep increasing). In this case, for p_1 and p_2 there is always the set 34 in FAULTY such that ALIVE $\cup$ 34 $= \Pi$ and p_1 and p_2 are not in 34. Thus, processes p_1 and p_2 take infinitely many simulation steps. But this is not the case for p_4. Process p_4 takes only finitely many simulation steps and eventually "backs-off" and stops simulating steps. Therefore, the simulated algorithm is executed as if the faulty set is 34 and thus has to terminate (since $34 \in \mathscr{A}$ and the algorithm is known to work against $\mathscr{A}$). If one of processes p_1 and p_2 has terminated, process p_4 can simply adopt the decision value and also terminate.

For the actual proof, let $b^* \in \mathscr{B}$ be the set of faulty processes in some run and assume that $D(\mathscr{A}, \mathscr{B})$. During the proof, it is important to keep in mind that there is a difference between process steps and simulation steps, i.e., the steps of the simulated algorithm the processes execute.

Lemma 3 *Algorithm 1 is well-defined.*

Proof To show that Algorithm 1 is well-defined, we need to show that there is always an appropriate set that can be added to the set FAULTY in the "foreach"-loop. If the set POSSIBLY- FAULTY is empty, then the "foreach"-loop immediately terminates.

If the set POSSIBLY- FAULTY is non-empty, then for every $b \in$ POSSIBLY- FAULTY, some $a \in \mathscr{A}$ with $D(a, b)$ is chosen such that $\forall a' \in$ FAULTY: $a \supseteq a'$. If b is the smallest set in POSSIBLY- FAULTY, we simply have to choose some set a where $D(a, b)$. Such a set has to exist, since we assume that $D(\mathscr{A}, \mathscr{B})$. If b is not the smallest set in POSSIBLY- FAULTY, then there exists a set a' that has been added to FAULTY in the

```
1  shared variable: STEPC_i := 0;                                          /* a SWMR register */
2  local variable: LASTSIMSTEPS := [0, ..., 0];                  /* the state at the last simulated step */
3  local variable: POSSIBLY- FAULTY := 𝔅;          /* the sets of processes that are possibly faulty */
4  local variable: FAULTY := ∅;          /* the sets of processes that are currently considered faulty */
5  while true do
6      if some other process has decided then adopt its decision value and halt;
7      let p_{i_1}, ..., p_{i_n} be the processes ordered by increasing STEPC (ties broken deterministically);
8      POSSIBLY- FAULTY := {∅, {p_{i_1}}, {p_{i_1}, p_{i_2}}, ..., {p_{i_1}, ..., p_{i_{n-1}}}} ∩ 𝔅;
9      FAULTY := ∅;
10     foreach b ∈ POSSIBLY- FAULTY, ordered by inclusion do
11         add some a ∈ 𝒜 to FAULTY such that D(a, b) and ∀a' ∈ FAULTY, a ⊇ a' (choose deterministically);
12     ALIVE := {p_j | STEPC_j > LASTSIMSTEPS[j]};
13     if ∃a ∈ FAULTY such that ALIVE ∪ a = Π and p_i ∉ a then
14         execute a step of A;
15         if decided then write decision value into special register and halt;
16         LASTSIMSTEPS := [STEPC_1, ..., STEPC_n];
17     STEPC_i := STEPC_i + 1;
```

Algorithm 1 The conservative back-off simulation for process p_i and $D(\mathscr{A}, \mathscr{B})$

previous step for some $b' \in$ POSSIBLY- FAULTY with $b \supsetneq b'$ (because the sets in POSSIBLY- FAULTY and by construction also the sets in FAULTY are ordered by inclusion). But this can only happen if $D(a', b')$. Thus, by Lemma 1 there has to exist a set $a \supseteq a'$ with $D(a, b)$ for our $b \supsetneq b'$. □

Lemma 4 *If no correct process halts, then eventually, for all sets* POSSIBLY- FAULTY *at a correct process:*

(i) $b^* \in$ POSSIBLY- FAULTY.
(ii) *if* $b \in$ POSSIBLY- FAULTY *at some process and* $b \subseteq b^*$, *then* $b \in$ POSSIBLY- FAULTY *at all correct processes.*

Proof Recall that b^* is the set of faulty processes in the run. Thus, all step counters at processes in set b^* change only finitely often. Furthermore, if no correct process halts, then all the other step counters increase infinitely often. Therefore, eventually b^* is always in POSSIBLY- FAULTY. Moreover, all the sets b that are in POSSIBLY- FAULTY with $b \subseteq b^*$ are the same at all correct processes, because they eventually always read the same counter-values. □

Lemma 5 *If no correct process halts, then there exists a set* $a^* \in \mathscr{A}$ *with* $a^* \supseteq b^*$ *such that:*

(i) a^* *is eventually always added to* FAULTY *in every iteration of the outer loop at every correct process and*
(ii) *for every* $a \subseteq a^*$ *that is infinitely often added to* FAULTY *at some correct process,* a *is infinitely often added to* FAULTY *at every correct process and*
(iii) *for every set* $a \supseteq b^*$ *that is infinitely often added to the set* FAULTY *at some correct process:* $a^* \subseteq a$.

Proof By Lemma 4, eventually at all correct processes, all subsets of b^* in POSSIBLY- FAULTY are the same. Thus, all the sets a that are added to FAULTY in the "foreach"-loop are the same at all correct processes up to and including the step where b^* is examined. When b^* is examined, by construction, the set a that is added to FAULTY satisfies $D(a, b^*)$ and thus $a \supseteq b^*$. Therefore, there is at least one set a with $a \supseteq b^*$ that is eventually in all sets FAULTY at all correct processes. Let a^* be the minimal such set. Then, (i) is true.

Moreover, eventually the sets a that are added to FAULTY before a^* are the same at all correct processes, because the "foreach"-loop operates ordered by inclusion. Thus (ii) is true.

To show (iii), assume that it is not true, i.e., there exists some set a with $a \supseteq b^*$ that is infinitely often added to FAULTY at some correct process and $a^* \not\subseteq a$. Then, by construction of FAULTY and (i): $a^* \supsetneq a$, because new sets are added to FAULTY only if they are supersets of all previously added. From (ii) follows that a is infinitely often added to FAULTY at all correct processes. But since a^* is chosen minimal, this implies that $a \not\supseteq b^*$. A contradiction. □

Lemma 6 *For every process that takes infinitely many simulation steps, eventually* ALIVE $\subseteq \Pi \backslash b^*$ *(i.e., eventually only correct processes are in* ALIVE*).*

Proof For a process that takes infinitely many simulation steps, LASTSIMSTEPS is updated infinitely often. Therefore, eventually only processes whose step-counters are infinitely often increased can be in ALIVE at such a process. But only the step-counters of correct processes (i.e. the processes in $\Pi \backslash b^*$) are increased infinitely often. □

Lemma 7 *If no correct process halts, then for every correct process that takes only a finite number of simulation steps, we eventually have* $\Pi \backslash b^* \subseteq$ ALIVE *(i.e., eventually all correct processes are in* ALIVE*).*

Proof Assume some process $p_j \notin b^*$ simulates only a finite number of steps of A. Then, eventually the content of LASTSIMSTEPS at p_j is constant, while the step-counters of the processes in $\Pi \backslash b^*$ keep constantly increasing, because we assume that none of them halts. Thus, eventually $\Pi \backslash b^* \subseteq$ ALIVE. □

Lemma 8 *If no correct process halts, then there exists a set $a^* \in \mathscr{A}$ of processes such that $a^* \supseteq b^*$ and exactly the processes in $\Pi \backslash a^*$ simulate infinitely many steps of A.*

Proof Let $a^* \in \mathscr{A}$ be the set from Lemma 5.

We first show by contradiction, that processes in a^* simulate only finitely many steps of A. Assume some process $p_j \in a^*$ simulates infinitely many steps of A. Then, at process p_j there exists infinitely often some set $a \in$ FAULTY such that ALIVE $\cup a = \Pi$ and $p_j \notin a$. This implies that $\Pi \backslash a \subseteq$ ALIVE. From Lemma 6 follows that eventually ALIVE $\subseteq \Pi \backslash b^*$ at p_j. By the transitivity of "$\subseteq$": $\Pi \backslash a \subseteq \Pi \backslash b^*$ and thus $b^* \subseteq a$. Therefore, by Lemma 5: $a^* \subseteq a$ and $p_j \notin a$. A contradiction to the fact that $p_j \in a^*$ and thus processes in a^* simulate only finitely many steps of A.

It remains to show that all processes in $\Pi \backslash a^*$ simulate infinitely many steps of A. Assume the contrary, i.e., some process p_j that is not in a^* simulates only finitely many steps of A. Since $a^* \supseteq b^*$, every such process is correct and, as no correct process halts, it takes infinitely many steps. By claim (i) of Lemma 5, a^* is eventually always in FAULTY at p_j and by Lemma 7, eventually ALIVE $\supseteq \Pi \backslash b^*$ at p_j. But then, ALIVE $\cup\, b^* = \Pi$ and thus ALIVE $\cup\, a^* = \Pi$ at p_j. A contradiction to the fact that p_j simulates only finitely many steps of A. Therefore, exactly the processes not in a^* simulate infinitely many steps. □

Theorem 2 *If $D(\mathscr{A}, \mathscr{B})$, then $\mathscr{A} \succcurlyeq \mathscr{B}$.*

Proof We show that Algorithm 1 decides in all runs of $\mathscr{B}$ for any colorless task T and for any algorithm A that solves T against $\mathscr{A}$. For this, it is sufficient if the simulation of A decides for any correct process, because A solves T against $\mathscr{A}$ and T is a colorless task. Thus, every other process can decide on the decision value of any other decided process.

Assume for contradiction that the simulation of A decides for no correct process, i.e., no correct process halts. From Lemma 3 follows that Algorithm 1 is well-defined and from Lemma 8 follows that there exists a set $a^* \in \mathscr{A}$ of processes such that exactly the set of processes in $\Pi \backslash a^*$ simulate infinitely many steps of A. Since $a^* \in \mathscr{A}$, every such run is indistinguishable from a real run of A against $\mathscr{A}$ and A has to terminate. Since we assume that $\Pi \notin \mathscr{A}$, i.e., that an adversary cannot crash all processes, we have $a^* \subsetneq \Pi$ and thus at least one correct process is not in a^* and will decide. A contradiction.

Thus, at least one correct process decides and all other correct processes can adopt its decision value. □

From this Theorem follows, that if $D(\mathscr{A}, \mathscr{U}_k)$, then k-set agreement cannot be implemented against $\mathscr{A}$, since it is impossible in $\mathscr{U}_k$ [3,12,15].

Corollary 1 *If $D(\mathscr{A}, \mathscr{U}_k)$, then k-set agreement cannot be implemented against $\mathscr{A}$.*

5 k-Set agreement protocol (necessary condition)

In this section, we show that if for adversaries $\mathscr{A}$ and $\mathscr{U}_k$ we have $\neg D(\mathscr{A}, \mathscr{U}_k)$, then k-set agreement can be implemented against $\mathscr{A}$. By the contrapositive, we get a necessary condition for the impossibility of implementing k-set agreement, namely if k-set agreement cannot be implemented against $\mathscr{A}$, then $D(\mathscr{A}, \mathscr{U}_k)$.

We compare an adversary $\mathscr{A}$ with $\mathscr{U}_k$, the k-failure adversary which contains all sets of size less or equal to k. We will show how to implement failure detector k-anti-Ω, which is known to be sufficient to implement k-set agreement against any adversary [17]. Basically, k-anti-Ω outputs, whenever queried, at least $n-k$ processes, such that at least one correct process is output only finitely often. Algorithm 2 implements k-anti-Ω against $\mathscr{A}$ under the assumption that $\neg D(\mathscr{A}, \mathscr{U}_k)$.

The key to the implementation is to find a set $u^* \in \mathscr{U}_k$ such that u^* contains at least one correct process, i.e., if the actual set of faulty processes is $a^* \in \mathscr{A}$, then $u^* \not\subseteq a^*$. It is sufficient though, that we eventually always find supersets of u^* of size at most k. The output for k-anti-Ω is then just the complement of these sets.

As in the previous section, we first try to narrow down the possibilities for the actual faulty set a^*. This is again achieved by using step-counters (denoted STEPC). The current estimates are stored in POSSIBLY- FAULTY. Then, we deterministically select a set $u_{init} \in \mathscr{U}_k$ that is not dominated by any $a \in \mathscr{A}$ (since $\neg D(\mathscr{A}, \mathscr{U}_k)$, there has to exist at least one). Although this set is not dominated by any a, it may contain no correct process (in particular, u_{init} may be the empty set). However, if so, i.e., if $u_{init} \subseteq a^*$, then by Lemma 2 there has to exist a strict superset of u_{init} which is not dominated by any $a \in \mathscr{A}$ with $a \supseteq u_{init}$ and we can select the maximal such set. Since a^* is eventually always in POSSIBLY- FAULTY, we eventually always choose the same $b^* \not\subseteq a^*$ as EST- u in the step for a^*. Although the supersets of a^* in POSSIBLY- FAULTY may differ in each round, our estimate will eventually always contain u^*, because the prefix until a^* in POSSIBLY- FAULTY is stable.

Consider Example 2 with $n = 3, k = 2, \mathscr{A} = \{\emptyset, 2, 12, 13, 23\}$ and $\mathscr{U}_2$ and recall that $\neg D(\mathscr{A}, \mathscr{U}_2)$. Then, for example we can choose $u_{init} = 3$, because for all $a \in \mathscr{A}$: $\neg D(a, 3)$. Thus EST- u is initially set to 3.

Assume first that p_2 is the only faulty process, i.e., that the actual faulty-set is 2. Eventually POSSIBLY- FAULTY can only

be $\{\emptyset, 2, 12\}$ or $\{\emptyset, 2, 23\}$. In any case, if $a = \emptyset$ is considered in the "foreach"-loop, then EST- u remains 3. If $a = 2$ is considered, then EST- u will be selected as 13, because EST- $u = 13$ is maximal such that $2 \not\supseteq$ EST- u. If $a = 12$ or $a = 23$ is considered, then EST- u remains still 13 and thus the emulated output of k-anti-Ω does not contain the correct processes p_1 and p_3, i.e., it will eventually be p_2.

Assume now that only process p_2 is correct, i.e., that the actual faulty-set is 13. Eventually POSSIBLY- FAULTY can only be $\{\emptyset, 13\}$. Thus, since we assume that EST- u is initialized with 3, the only choice for the next EST- u is 23. Therefore, eventually there is a correct process (process p_2) that is not in the output of k-anti-Ω.

If all processes are correct i.e., the faulty-set is $\emptyset$, then we have to avoid the possibility that the output alternates between 1, 2 and 3. Eventually POSSIBLY- FAULTY can be $\{\emptyset, 12\}$, $\{\emptyset, 13\}$, $\{\emptyset, 23\}$, $\{\emptyset, 2, 12\}$ or $\{\emptyset, 2, 23\}$. In any case, if $a = \emptyset$ is considered, then EST- u remains 3. After that, EST- u can be augmented, but 3 will eventually never be in the k-anti-Ω output. Therefore, eventually there is a correct process (process p_3) that is not in the output of k-anti-Ω.

For the actual proof, let $a^* \in \mathscr{A}$ be the actual set of faulty processes in some run and assume that $\neg D(\mathscr{A}, \mathscr{U}_k)$.

Lemma 9 *Algorithm 2 is well-defined.*

Proof To show that Algorithm 2 is well-defined, we first note that the sets in POSSIBLY- FAULTY can be ordered by inclusion.

We have to show that there exists some adequate u_{init}. Since it holds that $\neg D(\mathscr{A}, \mathscr{U}_k)$, there exists some $u \in \mathscr{U}_k$ such that $\forall a \in \mathscr{A} : \neg D(a, u)$. Thus, this u can be chosen as u_{init}. The set POSSIBLY- FAULTY will always be a subset of $\mathscr{A}$ and is ordered by inclusion. Therefore, $\neg D(a, u_{init})$ holds for all possible a in POSSIBLY- FAULTY. It is always possible to choose an appropriate new EST- u, because if EST- u is changed during the "foreach"-loop, then it is replaced by another EST- $u \in \mathscr{U}_k$ with $\neg D(a,$ EST- $u)$ for all possible subsequent a in POSSIBLY- FAULTY. □

Lemma 10 *Eventually always at all correct processes, for all sets* POSSIBLY- FAULTY:

(i) $a^* \in$ POSSIBLY- FAULTY.
(ii) *if* $a \in$ POSSIBLY- FAULTY *at some process and* $a \subseteq a^*$, *then* $a \in$ POSSIBLY- FAULTY *at all correct processes.*

Proof The proof is analogous to the proof of Lemma 4. □

Lemma 11 *Let* $a_1 \subsetneq \cdots \subsetneq a_l$ *be the sequence of sets in* POSSIBLY- FAULTY *at some time. Furthermore, let* EST- u_1, $\ldots$, EST- u_l *be the sequence of corresponding sets selected in the "foreach"-loop. Then for all* $1 \leq i \leq l$:

EST- $u_i \not\subseteq a_i$

Proof The proof is by contradiction. Assume EST- $u_i \subseteq a_i$ for some i. We have by definition of u_{init} and by the way EST- u_i is chosen in the "foreach"-loop that $\neg D(a_i,$ EST- $u_i)$. By Lemma 2, this implies that there exists some $u \in \mathscr{U}_k$, $u \supsetneq$ EST- u_i such that $\forall a' \in \mathscr{A}, a' \supseteq a_i : \neg D(a', u)$. But this contradicts the fact that EST- u_i is chosen maximal. □

Lemma 12 *There exists a set* $u^* \in \mathscr{U}_k$ *with* $a^* \not\supseteq u^*$ *that is eventually always a subset of* EST- u *in Line* 10 *at all correct processes.*

Proof By Lemma 11, at every step of the "foreach"-loop: EST- $u \not\subseteq a$ and by Lemma 10, eventually a^* is always contained in POSSIBLY- FAULTY. Thus, at every correct process, there exists a set $u^* \in \mathscr{U}_k$ with $a^* \not\supseteq u^*$ that is a subset of EST- u in the "foreach"-loop where a^* is considered. Let u^* be the minimal such set. To determine EST- u, only the previously considered sets in POSSIBLY- FAULTY are considered (it is otherwise deterministic). By Lemma 10, the sets a in POSSIBLY- FAULTY with $a \subseteq a^*$ are eventually the same at all correct processes, and the "foreach"-loop operates ordered by inclusion. Thus, every correct process will eventually always have u^* as a subset of EST- u. □

Theorem 3 *For all* $\mathscr{A}$, *if* $\neg D(\mathscr{A}, \mathscr{U}_k)$, *then it is possible to implement* k-*anti-*Ω *against* $\mathscr{A}$.

Proof By Lemma 9, Algorithm 2 is well-defined and from Lemma 12 follows that there exists a set $u^* \in \mathscr{U}_k$ with $a^* \not\supseteq u^*$ that is eventually in Line 10 always a subset of EST- u at all correct processes. Therefore, there exists a process $p \in u^*$ which is not in a^*. Thus, p is correct and p will eventually never be in the emulated failure detector output. This means, that the properties of k-anti-Ω are fulfilled (k-anti-Ω outputs, whenever queried, at least $n - k$ processes, such that at least one correct process is output only finitely often). Note that it is always possible to choose $n - k$ processes that are not in EST- u, because EST- $u \in \mathscr{U}_k$, i.e., $|$EST- $u| \leq k$.

In this way, we can emulate failure detector k-anti-Ω against $\mathscr{A}$ if $\neg D(\mathscr{A}, \mathscr{U}_k)$. □

Since k-anti-Ω is sufficient to implement k-set agreement against any adversary [17], we get:

Corollary 2 *If* $\neg D(\mathscr{A}, \mathscr{U}_k)$, *then it is possible to implement* k-*set agreement against* $\mathscr{A}$.

Furthermore:

Corollary 3 *If it is not possible to implement* k-*set agreement against* $\mathscr{A}$, *then* $D(\mathscr{A}, \mathscr{U}_k)$.

If we gather together Corollarys 1 and 2, we obtain a necessary and sufficient condition in terms of a structural predicate under which k-set agreement can be solved against an adversary.

1 shared variable: STEPC$_i$:= 0; /* a SWMR register */
2 local variable: u_{init} := deterministically select some maximal $u_{init} \in \mathcal{U}_k$ such that for all $a \in \mathcal{A}$, $\neg D(a, u_{init})$;
3 local variable: EST- $u := u_{init}$; /* the set of processes that are currently considered faulty */
4 local variable: POSSIBLY- FAULTY := $\mathcal{A}$; /* the sets of processes that are possibly faulty */
5 **while** *true* **do**
6 let $p_{i_1}, \ldots, p_{i_n}$ be the processes ordered by increasing STEPC (ties broken deterministically);
7 POSSIBLY- FAULTY := $\{\emptyset, \{p_{i_1}\}, \{p_{i_1}, p_{i_2}\}, \ldots, \{p_{i_1}, \ldots, p_{i_{n-1}}\}\} \cap \mathcal{A}$;
8 EST- $u := u_{init}$;
9 **foreach** $a \in$ POSSIBLY- FAULTY, *ordered by inclusion* **do**
10 EST- u := deterministically select some maximal $u' \in \mathcal{U}_k$ such that $u' \supseteq$ EST- u and $\forall a' \in \mathcal{A}, a' \supseteq a$ implies $\neg D(a', u')$;
11 STEPC$_i$:= STEPC$_i$ + 1;
12 output $n - k$ processes that are not in EST- u;

Algorithm 2 Implementation of k-anti-Ω for process p_i and $\neg D(\mathcal{A}, \mathcal{U}_k)$.

Theorem 4 *k-set agreement can be implemented against $\mathcal{A}$ if and only if $\neg D(\mathcal{A}, \mathcal{U}_k)$.*

We can now directly derive the disagreement power of an adversary by our structural predicate:

Theorem 5 *$dis(\mathcal{A}) = k$ if and only if $D(\mathcal{A}, \mathcal{U}_k) \wedge \neg D(\mathcal{A}, \mathcal{U}_{k+1})$.*

Together with Theorem 1, this induces the following Corollary:

Corollary 4 *For every k, there is no adversary $\mathcal{A} \subsetneq \mathcal{U}_k$ with $dis(\mathcal{A}) = k$.*

Proof Consider any adversary $\mathcal{A} \subsetneq \mathcal{U}_k$. From Theorem 1 follows that $\neg D(\mathcal{A}, \mathcal{U}_k)$. By Theorem 5: $dis(\mathcal{A}) \neq k$. □

In this sense, an adversary $\mathcal{U}_k$ is a minimal adversary with disagreement power k. Nevertheless, there may be other "locally minimal" adversaries with disagreement power k. For example, the 3-process adversary $\mathcal{A} = \{\emptyset, 1, 23\}$ is minimal for disagreement power 1 in the sense that every strict subset of $\mathcal{A}$ has disagreement power 0, i.e., it cannot even prevent consensus.

6 Equivalence classes

In this section we show that if two adversaries have the same disagreeement power, then they solve exactly the same set of colorless tasks: $dis(\mathcal{A}) = dis(\mathcal{B})$ if and only if $\mathcal{A} \succcurlyeq \mathcal{B}$ and $\mathcal{B} \succcurlyeq \mathcal{A}$.

We will start by showing how to use k-anti-Ω to simulate a set of k processes, such that at least one of the simulated processes takes infinitely many steps (this simulation, although seemingly simple, may be of independent interest). Our simulation in Algorithm 3 shows that every colorless task that can be solved against $\mathcal{U}^k_{k-1}$ can be solved against any adversary $\mathcal{A}$ against which k-anti-Ω is implementable.

We use the fact that there exists an implementation of consensus using Ω (the failure detector that eventually always outputs the same correct process everywhere) such that agreement and validity of consensus are satisfied, even if the Ω failure detector is bogus (i.e., it outputs infinitely often a faulty process or does not stabilize on one correct process) [14]. Furthermore, it is possible to extract an array of k failure detectors (denoted $\Omega'_1, \ldots, \Omega'_k$) from k-anti-Ω with the property that at least one of them is a "real" Ω and the rest may be bogus [17]. Thus, we can build consensus-like objects with every Ω' [14] and we have the property that at least one of these consensuses terminates. An object associated with a bogus Ω' may never terminate, but validity and agreement are never violated.

As our algorithm itself is relatively simple, we will only give the main points of the proof (compare e.g. [4] for a more detailed proof of a simulation).

We denote by consensus$_{j,r}$ the r-th instance of the consensus object built using Ω'_j. Furthermore, we associate with every $1 \leq j \leq k$ a "virtual" process q_j and all processes use the consensus$_{j,r}$-objects to agree on the simulated steps of q_j.

Without loss of generality, we assume that the algorithm that implements the task against $\mathcal{U}^k_{k-1}$ uses only one single-writer multiple-reader (SWMR) register per process. Three types of steps need to be considered:

- a *write*(v)-step in which a process writes v to its associated SWMR register,
- a *read*(q_j)-step in which a process reads the SWMR register associated to process q_j
- and an internal step which does not involve any registers

Note that these assumptions do not restrict the set of solvable tasks [16]. In our simulation, we will use snapshots to take atomic snapshots of all registers. Such atomic snapshots can be implemented using registers [1]. We assume that associated to each process p_i is an array of registers $R[i, 1], \ldots, R[i, k]$, one for each simulated process q_j. Every process can write to each of its associated registers.

A snapshot returns the state of all registers of all processes. All operations are assumed to take effect instantaneously.

The simulation works as follows: each process tries to simulate the steps of the k "virtual" processes $q_1, \ldots, q_k$ at its own rate. At the beginning, all processes propose their initial values to all k consensuses in parallel. Since the algorithm is deterministic, the internal steps of q_j can just be executed. To simulate the write-steps of q_j, every simulator p_i writes the value to be written together with the number of the currently simulated step to the shared register $R[i, j]$. To simulate a read-step of a register associated with a process q_j, a process takes a snapshot of all other shared registers associated with q_j and returns the "freshest" value (i.e., the value associated with the maximal step-number). Then, it proposes this value to the consensus corresponding to q_j and returns the result for the read-operation. In this way, it is ensured that all simulators will return exactly the same values for every q_j and thus all processes will simulate exactly the same steps. If some virtual process q_j has decided, the simulator just adopts that value and halts.

Theorem 6 *For every adversary $\mathscr{A}^n$ built upon a set of n processes and for every $k < n$: if $\neg D(\mathscr{A}^n, \mathscr{U}_k^n)$, then $\mathscr{U}_{k-1}^k \succcurlyeq \mathscr{A}^n$.*

Proof We assume an algorithm that solves a colorless task against $\mathscr{U}_{k-1}^k$ and use Theorem 3 to extract k-anti-Ω from $\mathscr{A}^n$ and thus are able to simulate a consensus-object for every one of the k processes q_j (of which some may be non-terminating). Since there is some j such that Ω'_j eventually always contains the same correct process, all simulated consensus$_{j,r}$-objects will terminate for every r. Thus, we can construct a consensus$_{j,r}$-object for every simulated read-step r and all correct processes simulate infinitely many steps of q_j.

Furthermore, since the execution of the simulated algorithm depends only on the values read (i.e., the algorithm is deterministic), for all j, all processes execute exactly the same steps for every virtual process q_j. By the definition of colorless tasks, it is allowed that any process picks up any other processes' input and output value and particularly, it is allowed that several processes have the same input or output values. It remains to show that every run of the virtual processes is indeed a run of the simulated algorithm in $\mathscr{U}_{k-1}^k$, i.e., it is indistinguishable from a real run. For this, we need to show that the sequence of the simulated operations on the registers is linearizable. This means, that there needs to exist some "point of linearization" for every distributed "write" and "read"-event at which the event appears to take effect instantaneously and the sequence of these events needs to be some legal execution according to the register specification.

For this, we use the fact that the sequence of operations on the real registers is linearizable. For the operations of our simulated registers, we define the points of linearization of the "write" and "read"-steps as follows. For the r_j-th step of a simulated process q_j: if the simulated step is a "read(q_x)"-step, then let v be the result of the corresponding execution of consensus$_{j,r_j}$ and let p_i be the process that first took a snapshot containing v among the processes that proposed v to consensus$_{j,r_j}$. Then, the point of linearization is exactly the point of linearization of this real "snapshot"-event. If the simulated step is a "write(v)"-step, then the point of linearization is exactly the point of linearization of the first real corresponding "write(v, r_j)"-step at some simulating process. Note that all processes will write the same value v in the r_j-th step of q_j, because the simulated algorithm depends only on the values read and these are the same at all processes for every process q_j, because of the consensuses.

This linearization behaves according to the register specification: no value is read before it is written and every read returns the latest written value. Thus, it respects the order of the linearization of the corresponding "real" events, because it always returns the value with associated maximal step-number r. Thus, every simulated run is indistinguishable from a real run and there exists at least one virtual process that eventually terminates and every correct simulator decides. □

Before we state our main result, we recall a theorem from [4] that, in our notation, states the following:

Theorem 7 (BG [4]) *For all n, for all k with $n > k$: $\mathscr{U}_k^n \succcurlyeq \mathscr{U}_k^{k+1}$.*

If we put all our other theorems together, we get the following result:

Theorem 8 *For any n and any two adversaries $\mathscr{A}^n$ and $\mathscr{B}^n$: $dis(\mathscr{A}^n) = dis(\mathscr{B}^n)$ if and only if $\mathscr{A}^n \succcurlyeq \mathscr{B}^n$ and $\mathscr{B}^n \succcurlyeq \mathscr{A}^n$.*

Proof Let k be the disagreement power of $\mathscr{A}^n$. By Theorem 5: $D(\mathscr{A}^n, \mathscr{U}_k^n)$ and therefore with Theorem 2:

$$\mathscr{A}^n \succcurlyeq \mathscr{U}_k^n \tag{2}$$

Furthermore, since $\mathscr{B}^n$ has also disagreement power k, we have: $\neg D(\mathscr{B}^n, \mathscr{U}_{k+1}^n)$ (Theorem 5). By Theorem 6: $\mathscr{U}_k^{k+1} \succcurlyeq \mathscr{B}^n$. With the transitivity of "$\succcurlyeq$" and Theorem 7:

$$\mathscr{U}_k^n \succcurlyeq \mathscr{B}^n \tag{3}$$

If we put (2) and (3) together, then by transitivity: $\mathscr{A}^n \succcurlyeq \mathscr{B}^n$. We obtain $\mathscr{B}^n \succcurlyeq \mathscr{A}^n$ in the same way. □

Thus, we can reduce any adversary with disagreement power k to adversary $\mathscr{U}_k$:

Corollary 5 *For any n, any k and any adversary $\mathscr{A}^n$ with $dis(\mathscr{A}^n) = k$: $\mathscr{A}^n \succcurlyeq \mathscr{U}_k^n$ and $\mathscr{U}_k^n \succcurlyeq \mathscr{A}^n$.*

Proof By Proposition 1, $dis(\mathscr{U}_k^n) = k$ and by Theorem 8: $\mathscr{A}^n \succcurlyeq \mathscr{U}_k^n$ and $\mathscr{U}_k^n \succcurlyeq \mathscr{A}^n$. □

```
1  shared variables: R[i, j] := (⊥, 0)   (for 1 ≤ j ≤ k);
2  local variables: init_j, r_j   (for 1 ≤ j ≤ k);
3  foreach 1 ≤ j ≤ k in parallel do
4      init_j := consensus_{j,0}(initial value of p_i);
5      r_j := 1;
6      while q_j has not decided do      /* start simulating steps of q_j with initial value init_j */
7          if next step of q_j is a write(v)-step then
8              R[i, j] := (v, r_j);
9          else if next step of q_j is a read(q_x)-step then
10             take a snapshot of all registers R[y, x]   (1 ≤ y ≤ n and 1 ≤ x ≤ k);
11             select value v such that there exists an index y with R[y, x] = (v, r) and r is maximal;
12             return consensus_{j,r_j}(v) for the read;
13         else
14             take internal step of q_j;
15         r_j := r_j + 1;
16     decide on q_j's decision value; halt;
```

Algorithm 3 Emulation of k processes against $\mathscr{U}_{k-1}^{k}$ using k-anti-Ω. Algorithm for process p_i.

7 Concluding remarks

This paper presents a novel way to precisely characterize *adversaries*: the notion of *disagreement power*, i.e., the biggest integer k for which an adversary can prevent processes from agreeing on k values. This notion partitions the set of all adversaries into n distinct *equivalence* classes, one for every disagreement power. Any two adversaries with the same disagreement power solve exactly the same set of (colorless) tasks (Sect. 6). We believe that our result could be extended to colored tasks but this is subject to future work. The key obstacle for this is that in colored tasks, the decision value of one terminated process is not necessarily enough to compute the output values of the others.

At the heart of our partitioning lies our simulation between algorithms that tolerate different adversaries (Sect. 4). Interestingly, the simulation works also if we assume the existence of stronger objects than registers or even non-deterministic object types. Furthermore, the simulation (as well as our implementation of k-set agreement parametrized with an adversary in Sect. 5) remains correct even if the adversary parameter changes for some time before it contains the actual adversary, e.g., because of some kind of failure detection mechanisms.

Acknowledgments We want to thank the anonymous referees for helpful suggestions to improve the paper.

References

1. Afek, Y., Attiya, H., Dolev, D., Gafni, E., Merritt, M., Shavit, N.: Atomic snapshots of shared memory. J. ACM **40**(4), 873–890 (1993)
2. Ashwinkumar, B.V., Patra, A., Choudhary, A., Srinathan, K., Rangan, C.P.: On tradeoff between network connectivity, phase complexity and communication complexity of reliable communication tolerating mixed adversary. In: PODC, pp. 115–124 (2008)
3. Borowsky, E., Gafni, E.: Generalized FLP impossibility result for t-resilient asynchronous computations. In: STOC, pp. 91–100 (1993)
4. Borowsky, E., Gafni, E., Lynch, N.A., Rajsbaum, S.: The BG distributed simulation algorithm. Distrib. Comput. **14**(3), 127–146 (2001)
5. Chaudhuri, S.: More choices allow more faults: set consensus problems in totally asynchronous systems. Inf. Comput. **105**(1), 132–158 (1993)
6. Chandra, T.D., Hadzilacos, V., Jayanti, P., Toueg, S.: Generalized irreducibility of consensus and the equivalence of t-resilient and wait-free implementations of consensus. SIAM J. Comput. **34**(2), 333–357 (2004)
7. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. J. ACM **43**(4), 685–722 (1996)
8. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM **43**(2), 225–267 (1996)
9. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
10. Fitzi, M., Maurer, U.M.: Efficient byzantine agreement secure against general adversaries. In: DISC, pp. 134–148 (1998)
11. Herlihy, M., Rajsbaum, S.: The decidability of distributed decision tasks (extended abstract). In: STOC, pp. 589–598 (1997)
12. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6), 858–923 (1999)
13. Junqueira, F., Marzullo, K.: A framework for the design of dependent-failure algorithms. Concurr. Comput. Pract. Exp. **19**(17), 2255–2269 (2007)
14. Lo, W.-K., Hadzilacos, V.: Using failure detectors to solve consensus in asynchronous shared-memory systems (extended abstract). In: WDAG, pp. 280–295 (1994)
15. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: the topology of public knowledge. SIAM J. Comput. **29**(5), 1449–1483 (2000)
16. Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware (detailed abstract). In: FOCS, pp. 233–243 IEEE (1986)
17. Zielinski, P.: Anti-Omega: the weakest failure detector for set agreement. Distrib Comput **22**(5–6), 335–348 (2010)