The Disagreement Power of an Adversary

Carole Delporte-Gallet[1], Hugues Fauconnier[1],
Rachid Guerraoui[2], and Andreas Tielmann[1]

[1] LIAFA, Université Paris Diderot
Paris, France
[2] Distributed Programming Laboratory, EPFL
Lausanne, Switzerland

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 the remaining $(2^{2^n} - n)$ adversaries that might crash certain combination of processes and not others?
This paper presents a precise way to characterize such adversaries by introducing 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 how this notion enables to 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 deterministically in a shared memory model where t processes can crash is exactly $t+1$ [1–3]. 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 FLP [4].

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 $(2^{2^n} - n)$ adversaries that can crash certain subsets of processes of a certain size but not others of the same size? In particular, given any adversary $\mathcal{A}$, for what k does $\mathcal{A}$ prevent k-set agreement [5]? This paper addresses this question and derives from the answer equivalence classes between adversaries. More specifically, we characterize the power of an adversary $\mathcal{A}$ by the biggest k for which k-set agreement cannot be solved with $\mathcal{A}$, which we call here the *disagreement power* of $\mathcal{A}$. We show how to compute the disagreement power of

an adversary and we show that adversaries within the same class solve the same set of (colorless[3]) tasks.

Beyond intellectual curiosity, studying "non-uniform" adversaries might even be practically motivated by modern multicore architectures where the failures of processes in the same core might all be correlated [8–10].

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

We give a general characterization of adversaries that enables one to automatically compute their disagreement power. Namely, we introduce a *structure predicate* on adversaries, parameterized by an integer k, and which, intuitively, checks for any set of faulty processes of size less or equal k, whether there is some adequate superset in the 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 some k, then k-set agreement can be solved with the k-failures adversary which in turn is known to be impossible [1–3]. Hence, an adversary that satisfies the predicate has disagreement power at least k. We do this through 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 faulty in some set where the simulated algorithm is known to work. Conversely (necessary condition), we show how to solve k-set agreement with any adversary $\mathcal{A}$ that does not satisfy the predicate for some k. We do this by showing how to implement failure detector k-anti-Ω [11], known in turn 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. 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 which in turn can simulate every other k-failure adversary [6, 12]. This is technically achieved by implementing $(k+1)$-anti-Ω for the adversary and translating it to a vector of $k + 1$ Ω failure detectors [13] of

[3] Intuitively, in a colorless task [6, 7] any process can adopt any input or output value of any other process without violating the task specification.

[4] When appropriate, we will use e.g. 23 as shorthand for the set $\{2, 3\}$.

which at least one is a "real" Ω (i.e. it outputs eventually everywhere the same correct process). 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 adversary from known results from another 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 $(n-1)$-set agreement for the "wait-free" adversary, topology is not needed for all the other ones. Results concerning other k-failures ("uniform") adversaries can be deduced by [6, 12], whereas results for all other ("non-uniform") $(2^{2^n}-n)$ adversaries can be deduced from our characterization.

The remainder of the paper is structured as follows. We first define our model in Section 2. We then introduce our notion of disagreement power and our structural predicate in Section 3. We present our conservative back-off simulation and use it in Section 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 Section 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 Section 6 and conclude the paper with some general remarks in Section 7.

2 Model and Definitions

We assume systems 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 a set $\Pi = \{p_1, p_2, ..., p_n\}$ of n processes sharing atomic registers. Processes might crash. 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 $\mathcal{A}$ for which there is always at least one correct process, i.e. $\Pi \notin \mathcal{A}$.

A run of an algorithm A is an infinite sequence of steps of the processes. Given an adversary $\mathcal{A}$, associated with every run is a set of processes $a \in \mathcal{A}$ 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 make an infinity of steps in some run associated with a is then exactly $\Pi \setminus a$.

The classical n process k-failure adversary, denoted $\mathcal{B}_k^n$ is the adversary for which at most k ($0 \leq k \leq n-1$) processes may crash: $\mathcal{B}_k^n = \{b \subseteq \Pi \mid |b| \leq k \wedge |\Pi| = n\}$. Where the number of processes is clear from the context, we will omit the n (i.e. $\mathcal{B}_k = \mathcal{B}_k^n$).

Tasks. Generally, we say that algorithm A solves a task T in adversary $\mathcal{A}$ if every run of A with associated $a \in \mathcal{A}$ satisfies the specification of T (we say also A implements T for adversary $\mathcal{A}$). More specifically, a task is a tuple $(\mathcal{I}, \mathcal{O}, \Delta)$, where $\mathcal{I}$ is a set of vectors of input values and $\mathcal{O}$ 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 total relation from $\mathcal{I}$ to $\mathcal{O}$. Then, a task is solved if for input vector $I \in \mathcal{I}$, an output vector $O \in \mathcal{O}$ is computed such that $O \in \Delta(I)$.

In the following, we restrict ourselves to specific colorless tasks [6, 7]. Let $val(V)$ be the set of values in some vector V. A *colorless* task is such that if $O \in \Delta(I)$, then for every I' with $val(I') \subseteq val(I)$: $I' \in \mathcal{I}$ and $\Delta(I') \subseteq \Delta(I)$. Furthermore, for every O' with $val(O') \subseteq val(O)$: $O' \in \mathcal{O}$ and $O' \in \Delta(I)$. As a result, the specification of a colorless task is independent 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 any set of values with $|S| \geq k+1$. In k-set agreement, $\mathcal{I}$ and $\mathcal{O}$ are the sets of all vectors of values from S such that for all $O \in \mathcal{O}$, $|val(O)| \leq k$ and for every $I \in \mathcal{I}$: $O \in \Delta(I)$ iff $val(O) \subseteq val(I)$.

Consensus is 1-set agreement. k-set agreement can be solved in $\mathcal{B}_l$ iff $0 \leq l \leq k-1$ [1–3].

In one of our proofs, we will use a distributed oracle called k-anti-Ω [11]: 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. If $k = 1$, k-anti-Ω is equivalent to the eventual leader Ω failure detector, the weakest failure detector for consensus [13, 14]. If $k = n-1$, k-anti-Ω is anti-Ω, the weakest failure detector to solve $(n-1)$-set agreement [11].

3 Disagreement Power

We define the *disagreement power* of an adversary $\mathcal{A}$ to be the maximal k for which it is impossible to implement k-set agreement in $\mathcal{A}$. More precisely:

Definition 1. *An adversary $\mathcal{A}$ has* disagreement power *k, denoted $dis(\mathcal{A})$, if (1) it is impossible to implement k-set agreement in $\mathcal{A}$, and (2) it is possible to implement $(k+1)$-set agreement in $\mathcal{A}$.*

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

Proposition 1. $dis(\mathcal{B}_k) = k$

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 $\mathcal{A}$ is* stronger *than an adversary $\mathcal{B}$ (denoted $\mathcal{A} \succcurlyeq \mathcal{B}$) if every colorless task that can be solved in $\mathcal{A}$ can be solved in $\mathcal{B}$.*

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

$$(a \supseteq b) \quad \textit{and} \quad (\forall b' \in \mathcal{B}, b' \supsetneq b, \exists a' \in \mathcal{A}, a' \supseteq a : D(a', \mathcal{A}, b', \mathcal{B}))$$

In the base case, when there is no strict superset of b in $\mathcal{B}$, then this translates to $a \supseteq b$. Where $\mathcal{A}$ and $\mathcal{B}$ 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 $\mathcal{A}$* dominates *an adversary $\mathcal{B}$ (denoted $D(\mathcal{A}, \mathcal{B})$) if and only if the following property is satisfied:*

$$\forall b \in \mathcal{B}, \exists a \in \mathcal{A} : D(a, \mathcal{A}, b, \mathcal{B}).$$

This property is intricate. One may think that if for all $b_0 \subset b_1 \ldots \subset b_x$ in $\mathcal{B}$ there exist $a_0 \subseteq a_1 \ldots \subseteq a_x$ in $\mathcal{A}$ such that $b_i \subseteq a_i$ for all i then $D(\mathcal{A}, \mathcal{B})$. But this is not the case. Consider the following example:

Example 1. Assume $n = 3$ and consider two adversaries (we use $ij\ldots$ as a shorthand for the set $\{p_i, p_j, \ldots\}$):

$$\mathcal{A} = \{\emptyset, 2, 12, 13, 23\}$$
$$\mathcal{B}_2 = \{\emptyset, 1, 2, 3, 12, 13, 23\}$$

In this example for all $b_0 \subset b1 \subset b2$ there exist $a_0 \subseteq a1 \subseteq a2$. But $\neg D(\mathcal{A}, \mathcal{B}_2)$, because for all $a \in \mathcal{A}$, $\neg D(a, 3)$.

Example 2. Consider now a slightly different example with $n = 4$ and the following adversaries :

$$\mathcal{A} = \{\emptyset, 12, 34, 123, 124, 134, 234\}$$
$$\mathcal{B}_2 = \{\emptyset, 1, 2, 3, 4, 12, 13, 14, 23, 24, 34\}$$

In this example, $D(\mathcal{A}, \mathcal{B}_2)$, i.e. for every $b \in \mathcal{B}_2$ there exists an $a \in \mathcal{A}$ such that $D(a, b)$ (e.g. $D(\emptyset, \emptyset)$, $D(12, 2)$ and $D(124, 24)$).

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

Theorem 1. *Consider any k with $1 \leq k \leq n-1$ and any element $b \in \mathcal{B}_k$. Then $\neg D(\mathcal{B}_k \setminus \{b\}, \mathcal{B}_k)$.*

Proof. We show that for all $a \in \mathcal{B}_k \setminus \{b\}$, $\neg D(a, b)$. If $|b| = k$, then this is immediately clear, because there cannot be any superset of b in $\mathcal{B}_k \setminus \{b\}$. Otherwise, if $|b| < k$ then assume that $a \in \mathcal{B}_k$ is any set such that $D(a, b)$. Hence, there is some set $b' \supsetneq b$ with $|b'| = k$ such that the number of processes not in a is maximal in b'. Assume there exists some $a' \supseteq a$ and $a' \supseteq b'$. Then $|a'| \geq |a \cup b'| > k$, because $|b'| = k$ and there is at least one element in $a \setminus b'$. Thus $a' \notin \mathcal{B}_k$ and we have a contradiction.

4 The Conservative Back-off Simulation (Sufficient Condition)

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

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

The goal of the simulation is to identify, in every possible run with a set of faulty processes $b^* \in \mathcal{B}$, a set of processes $a^* \in \mathcal{A}$ with $b^* \subseteq a^*$ (i.e. more failures in a^* than in b^*). Hence, the processes outside a^* can use the given algorithm which is known to terminate for every $a^* \in \mathcal{A}$. 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.

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

In these steps, our property $D(\mathcal{A}, \mathcal{B})$ is needed. For every $b \in$ *possibly-faulty*, starting from the smallest, some $a \in \mathcal{A}$ with $D(a, b)$ that is a superset of all other elements in *faulty* is deterministically chosen and added to *faulty*. Since $D(a, b)$, and every next $b' \in$ *possibly-faulty* is a superset of b, it is guaranteed that in the following there will always be an $a' \in \mathcal{A}$ that is a superset of a and $D(a', b')$. 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 in b^* is such that $b^* \subseteq a$. Then, 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 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 the last own simulation step using the variable *lastsimsteps*. 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 2: $\mathcal{A} = \{\emptyset, 12, 34, 123, 124, 134, 234\}$, $\mathcal{B}_2$ and $n = 4$. If the actual faulty-set is 3, then eventually *possibly-faulty* can only be: $\{\emptyset, 3, 23\}$, $\{\emptyset, 3, 13\}$ or $\{\emptyset, 3, 34\}$, because process 3 takes the least number of steps. By construction, *faulty* will be $\{\emptyset, 34, 234\}$, $\{\emptyset, 34, 134\}$ or $\{\emptyset, 34, 234\}$ respectively. For the three processes p_1, p_2 and p_4 that take infinitely many steps, eventually $alive \subseteq 124$. If one of these processes takes only finitely many simulation steps, then $alive = 124$ at this process. 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 respectively p_2 are not in 34. But this is not the case for p_4. Thus, p_4 takes only finitely many steps and only processes p_1 and p_2 take infinitely many steps. Therefore, the simulated algorithm is executed as if the faulty set is 34.

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

```
1  Stepc_i := 0;                                          /* a SWMR register */
2  lastsimsteps_i := [0, ..., 0];      /* the state at the last simulated step */
3  while true do
4      if some other process has decided then adopt its decision value and decide;
5      let p_{i_1}, ..., p_{i_n} be the processes ordered by increasing Stepc (ties broken
       deterministically);
6      possibly-faulty_i := {∅, {p_{i_1}}, {p_{i_1}, p_{i_2}}, ..., {p_{i_1}, ..., p_{i_{n-1}}}} ∩ B;
7      faulty_i := ∅;
8      foreach b ∈ possibly-faulty_i, ordered by inclusion do
9          add some a ∈ A to faulty_i s.t. D(a, b) and ∀a' ∈ faulty_i, a ⊇ a' (choose
           deterministically);
10     alive_i := {p_j | Stepc_j > lastsimsteps_i[j]};
11     if ∃a ∈ faulty_i, alive_i ∪ a = Π and p_i ∉ a then
12         execute a step of Alg;
13         if decided then write decision value into special register;
14         lastsimsteps_i := [Stepc_1, ..., Stepc_n];
15     Stepc_i := Stepc_i + 1;
```

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

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

Let $b^* \in \mathcal{B}$ be the actual set of faulty processes in some run. Then, eventually always $b^* \in$ possibly-faulty and all $b \in$ possibly-faulty with $b \subseteq b^*$ are the same at all processes, because all the step counters at processes in sets $b \subseteq b^*$ change only finitely often and the step counters of some of the processes in all other sets increase infinitely often.

In the "for"-loop, for every $b \in$ possibly-faulty, some $a \in \mathcal{A}$ with $D(a, b)$ is chosen such that $\forall a' \in faulty_i$, $a \supseteq a'$. It is here where we need $D(\mathcal{A}, \mathcal{B})$. If b is the smallest set in possibly-faulty, we simply have to choose some set a where $D(a, b)$. In all following steps, the recursiveness of the domination predicate is needed. Let a' be the set that has been added to faulty in the previous step in $b' \in$ possibly-faulty. Thus $D(a', b')$ and we need in fact some $a \supseteq a'$ with $D(a, b)$ where $b \supsetneq b'$. And this follows immediately from $D(a', b')$.

Let $a^* \in \mathcal{A}$ be the smallest set with $a^* \supseteq b^*$ that is eventually always added to $faulty_i$. Such a set has to exist (e.g. the one that is added in the step where $b = b^*$). Then, eventually, and at all correct processes, for all sets $a \in faulty_i$ where $a \cup alive_i = \Pi$, a is a superset of a^*, because for all strict subsets a' of a^* there is at least one process $p \notin a'$ that makes only finitely many steps. Since eventually only processes that are not in such an a take steps, processes in a^* simulate only finitely many steps of Alg.

Assume some process p_j that is not in a^* simulates only finitely many steps of Alg. Since $a^* \supseteq b^*$, all these processes take infinitely many steps. Therefore, eventually, $alive_j \supseteq \Pi \setminus a^*$. But then, $alive_j \cup a^* = \Pi$. A contradiction to the fact that p_j simulates only finitely many steps of Alg. Therefore, exactly the processes not in a^* simulate infinitely many steps. Since $a^* \in \mathcal{A}$, Alg has to terminate.

From this Theorem follows, that if $D(\mathcal{A}, \mathcal{B}_k)$, then k-set agreement cannot be implemented in $\mathcal{A}$, since it is impossible in $\mathcal{B}_k$ [1–3].

Corollary 1. *If $D(\mathcal{A}, \mathcal{B}_k)$, then k-set agreement cannot be implemented in $\mathcal{A}$.*

5 k-Set Agreement Protocol (Necessary Condition)

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

We compare an adversary $\mathcal{A}$ with the k-failure adversary which contains all sets of size less or equal k. We show that if $\neg D(\mathcal{A}, \mathcal{B}_k)$, then it is possible to implement k-set agreement for $\mathcal{A}$. For this, it is sufficient to show how to

implement k-anti-Ω, since this is sufficient to implement k-set agreement in adversary $\mathcal{A}$ [11]. Basically, k-anti-Ω outputs, whenever queried, at least $n-k$ processes, s.t. at least one correct process is output only finitely often. Algorithm 2 implements k-anti-Ω.

The key to the implementation is to find a set b^* such that b^* contains at least one non-faulty process, i.e. if the actual set of faulty processes is a^*, then $b^* \not\subseteq a^*$. It is sufficient though, that we eventually always find supersets of b^* 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. The current estimations are stored in *possibly-faulty*. Then, we take the smallest set $b_{init} \in \mathcal{B}_k$ that is not dominated by any $a \in \mathcal{A}$ (since $\neg D(\mathcal{A}, \mathcal{B}_k)$, there has to exist at least one). Although this set is not dominated by any a, it may contain no correct process (in particular, b_{init} may be the empty set). However, if so, then by the recursive nature of the domination property, there has to exist a strict superset of b_{init} which is not dominated by any $a \in \mathcal{A}$ with $a \supseteq b_{init}$ (if $b_{init} = \emptyset$, then this applies to all $a \in \mathcal{A}$). By an iterated use of this property, for every possible $a \in$ *possibly-faulty*, the inner "while"-loop ends. Thus, for all $a \in$ *possibly-faulty*: $a \not\supseteq$ *est* for the corresponding *est* after the loop. Since a^* is eventually always in *possibly-faulty*, we eventually always choose the same $b^* \not\subseteq a^*$ in the step for a^*. Although the supersets of a^* in *possibly-faulty* may differ in each round, our estimate will eventually always contain b^*, because some prefix in *possibly-faulty* is stable.

Consider Example 1 with $n = 3$ and $k = 2$: $\mathcal{A} = \{\emptyset, 2, 12, 13, 23\}$ and $\mathcal{B}_2$ and recall that $\neg D(\mathcal{A}, \mathcal{B}_2)$. Then, for example $b_{init} = 3$ and thus *est* is initially set to 3.

Assume first that the actual faulty-set is 1. Eventually *possibly-faulty* will be $\{\emptyset, 12\}$ or $\{\emptyset, 13\}$. In any case, if $a = \emptyset$ is considered, then *est* remains 3. If $a = 12$ is considered, then *est* remains 3 and thus the failure detector output does not contain 3.

Assume now that all the processes are correct i.e. the faulty-set is $\emptyset$. We have to avoid, in this case, that the output alternates between 1, 2 and 3. Eventually *possibly-faulty* will be $\{\emptyset, 1, 12\}$, $\{\emptyset, 1, 13\}$, $\{\emptyset, 2, 12\}$, $\{\emptyset, 2, 23\}$, $\{\emptyset, 3, 13\}$ or $\{\emptyset, 3, 23\}$. In any case, if $a = \emptyset$ is considered, then *est* remains 3. After that, *est* can be augmented, but 3 will eventually never be in the ouput of k-anti-Ω. Therefore, eventually there is a correct process (3) that is not in the output of k-anti-Ω.

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

Proof. If $\neg D(\mathcal{A}, \mathcal{B}_k)$, then:

$$\exists b \in \mathcal{B}_k, \forall a \in \mathcal{A}, \neg D(a, b).$$

Thus, this b can be chosen as b_{init}. If a does not dominate b for $\mathcal{A}$ and $\mathcal{B}_k$, then

$$(a \not\supseteq b) \quad \vee \quad (\exists b' \in \mathcal{B}_k, b' \supsetneq b, \forall a' \in \mathcal{A}, a' \supseteq a, \neg D(a', b')). \tag{1}$$

Algorithm 2: Implementation of k-anti-Ω

```
1  Stepc_i := 0;                                          /* a SWMR register */
2  b_init := some set in B_k s.t. for all a ∈ A, ¬D(a, b_init);
3  while true do
4      let p_{i_1}, ..., p_{i_n} be the processes ordered by increasing Stepc (ties broken
       deterministically);
5      possibly-faulty_i := {∅, {p_{i_1}}, {p_{i_1}, p_{i_2}}, ..., {p_{i_1}, ..., p_{i_{n-1}}}} ∩ A;
6      est_i := b_init;
7      foreach a ∈ possibly-faulty_i, ordered by inclusion do
8          while a ⊇ est_i do
9              est_i := determin. choose some b ∈ B_k, b ⊋ est_i s.t. ∀a' ∈ A, a' ⊇ a:
                   ¬D(a', b);
10     if |est_i| < k then add some processes to est_i until |est_i| = k
11     Stepc_i := Stepc_i + 1;
12     output Π \ est_i;
```

Let $a^* \in \mathcal{A}$ be the actual set of faulty processes in some run. Then, eventually, *possibly-faulty* contains a^* and all $a \in$ *possibly-faulty*, $a \subseteq a^*$ are the same at all processes, because all the step counters at these processes change only finitely often and the step counters of some of the processes in all supersets increase infinitely often.

Since $\neg D(\mathcal{A}, \mathcal{B}_k)$, there exists some good b_{init}. For every $a \in$ *possibly-faulty* in every step in the "for"-loop, $\neg D(a, est)$, because otherwise it would not have been chosen as b_{init} or in the inner "while"-loop. Thus, either $a \not\supseteq est$ and the inner "while"-loop immediately terminates, or it follows from (1), that there exists some some $b \in \mathcal{B}_k$, $b \supsetneq est_i$ s.t. $\forall a' \in \mathcal{A}, a' \supseteq a : \neg D(a', b)$, i.e. the loop continues. Since in every step of the inner "while"-loop, est grows and $est \in \mathcal{B}_k$, the loop ends after at most k steps.

Let $b^* \supseteq b_{init}$ be the maximal set such that b^* is eventually always a subset of est_i at the end of the "for"-loop. Since the prefix of the subsets of a^* is stable in *possibly-faulty*, $a^* \supseteq b^*$, because this is the terminating condition of the inner "while"-loop. Therefore, $a^* \not\supseteq b^*$ and there exists a process $p \in b^*$ which is not in a^* and the properties of k-anti-Ω are fulfilled.

Then, we get:

Corollary 2. *If k-set agreement cannot be implemented in $\mathcal{A}$, then $D(\mathcal{A}, \mathcal{B}_k)$.*

If we gather together Theorem 2 and Theorem 3, we obtain a necessary and sufficient condition in terms of structured predicate under which an adversary can solve the k-set agreement.

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

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 (1) $D(\mathcal{A}, \mathcal{B}_k)$, and (2) $\neg D(\mathcal{A}, \mathcal{B}_{k+1})$*

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 $\mathcal{B} \succcurlyeq \mathcal{A}$

Before showing that all adversaries with the same disagreement power solve the same set of (colorless) tasks, we show that the ability of an adversary to prevent agreement is independent of the number of processes (as long as $n \geq k+1$). After that, we show that for any two adversaries $\mathcal{A}$ and $\mathcal{B}$ with the same disagreement power k, $\mathcal{A} \succcurlyeq \mathcal{B}$.

6.1 Robustness against the Number of Processes

Before we state our results, we recall a theorem from [6] that, in our notation, states the following:

Theorem 6. *(BG [6]). For all n, for all k with $n > k$: $\mathcal{B}_k^n \succcurlyeq \mathcal{B}_k^{k+1}$.*

Theorem 7. *For every adversary $\mathcal{A}^n$ built upon a set of n processes, for every $k < n$: if k-set agreement for $k+1 \leq n$ processes can be implemented in $\mathcal{A}^n$, then k-set agreement can be implemented in $\mathcal{A}^n$.*

Proof. Assume k-set agreement can be implemented for $k+1$ processes in adversary $\mathcal{A}^n$. Assume for contradiction that $D(\mathcal{A}^n, \mathcal{B}_k^n)$. Then, with Theorem 2, k-set agreement for $k+1$ processes can be implemented in $\mathcal{B}_k^n$ and with Theorem 6 it follows that k-set agreement can be implemented in $\mathcal{B}_k^{k+1}$. A contradiction to [1–3]. Thus, $\neg D(\mathcal{A}^n, \mathcal{B}_k^n)$ which means by Theorem 3 that k-set agreement can be implemented in $\mathcal{A}^n$.

Thus, the ability of an adversary to prevent an agreement of k values is independent from the number of processes, i.e. if it cannot prevent agreement for $k+1$ processes, then it cannot prevent agreement for any $n > k$ processes. In this sense, the disagreement power is robust against the number of processes.

6.2 Simulating k Processes with k-anti-Ω

In the following, we will show 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). With the simulation, we can show that every colorless task that can be solved in $\mathcal{B}_{k-1}^k$ can be solved with any adversary $\mathcal{A}$ where k-anti-Ω is implementable. We use here the fact that it is possible to extract an array of k Ω-failure detectors $\Omega_1, \ldots, \Omega_k$ from k-anti-Ω with the property that at least one of them is a "real" Ω (i.e. it eventually outputs everywhere always the same correct process) [11].

Thus we can build k consensus objects [14], one for every Ω and we have the property, that at least one consensus terminates infinitely often. Note that for the consensuses associated with a bogus Ω (i.e. one that outputs infinitely often faulty processes or does not stabilize on one process), in [14], the agreement and validity properties of consensus are never violated.

We denote with $\text{consensus}_{j,r}$ the r-th invocation of the consensus object associated with Ω_j. Furthermore, we associate with every $1 \leq j \leq k$ a "virtual" process q_j and all processes use the $\text{consensus}_{j,r}$-objects to agree on the simulated steps of q_j.

Without loss of generality, we assume that the algorithm that implements the task in $\mathcal{B}^k_{k-1}$ 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 $read(p_j)$-step in which a process reads the SWMR register associated to process p_j
- and an internal step which does not involve any registers

These assumptions do not restrict the set of solvable tasks [15].

The simulation works as follows: 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 its own register $R[i, j]$. To simulate a read step of q_j, a process scans all other processes 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 all will simulate exactly the same steps. If some virtual process q_j has decided, the simulator just adopts that value and halts.

Theorem 8. *For every adversary $\mathcal{A}^n$ build upon a set of n processes, for every $k < n$, : if $\neg D(\mathcal{A}^n, \mathcal{B}^n_k)$, then $\mathcal{B}^k_{k-1} \succcurlyeq \mathcal{A}^n$.*

Proof. We assume an algorithm that solves a colorless task in $\mathcal{B}^k_{k-1}$ and use Theorem 3 to extract k-anti-Ω from $\mathcal{A}^n$ and thus create $\text{consensus}_{j,r}$-objects for every j and k. Since there is some j such that Ω_j contains eventually always a correct process, 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 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 $\mathcal{B}^k_{k-1}$, i.e. it is indistinguishable from a real run. For this, we need to show that the sequence of the simulated operations

Algorithm 3: Simulation of $\mathcal{B}^k_{k-1}$ for process p_i

```
1  for 1 ≤ j ≤ k do R[i, j] := (⊥, 0);
2  init_i := initial value;
3  foreach 1 ≤ j ≤ k in parallel do
4      init_j := consensus_{j,0}(init_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(p_x)-step then
10             select v s.t. r is max. ∀(v, r) where ∃y : R[y, x] = (v, r);
11             return consensus_{j,r_j}(v) for the read;
12         else
13             take internal step of q_j;
14         r_j := r_j + 1;
15     decide on q_j's decision value; halt;
```

on the registers is linearizable. But this follows from the fact that the sequence of the real registers is linearizable and every simulated operation corresponds to some operation of the real run.

Thus, every simulated run will eventually terminate at at least one virtual process and every simulator decides.

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

Theorem 9. *For any two adversaries $\mathcal{A}$ and $\mathcal{B}$: $dis(\mathcal{A}) = dis(\mathcal{B})$ if and only if $\mathcal{A} \succcurlyeq \mathcal{B}$ and $\mathcal{B} \succcurlyeq \mathcal{A}$*

Proof. Since $\mathcal{A}$ has disagreement power k, it is impossible to implement k-set agreement in $\mathcal{A}$. Thus $D(\mathcal{A}, \mathcal{B}_k)$ (Theorem 3) and therefore $\mathcal{A} \succcurlyeq \mathcal{B}_k$ (Theorem 2). Furthermore, since $\mathcal{B}$ has also disagreement power k, it is possible to implement $(k+1)$-set agreement in $\mathcal{B}$. Therefore, since it is impossible to implement $(k+1)$-set agreement in $\mathcal{B}_{k+1}$ [8–10]: $\neg D(\mathcal{B}, \mathcal{B}_{k+1})$ (Theorem 2). Thus, with Theorem 8, $\mathcal{B}^{k+1}_k \not\succcurlyeq \mathcal{B}$. With Theorem 6 (BG), $\mathcal{B}_k \not\succcurlyeq \mathcal{B}$. If we put all these results together, $\mathcal{A} \succcurlyeq \mathcal{B}$. We obtain $\mathcal{B} \succcurlyeq \mathcal{A}$ in the same way.

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 (Section 6). We believe that our result could be extended to colored tasks but this is subject to future work.

At the heart of our partitioning lies our simulation between adversaries (Section 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 with a given adversary in Section 5) remains correct even if the adversary is known only eventually, i.e., not necessarily from the beginning.

References

1. Herlihy, M., Shavit, N.: The topological structure of asynchronous computability. J. ACM **46**(6) (1999) 858–923
2. Borowsky, E., Gafni, E.: Generalized flp impossibility result for t-resilient asynchronous computations. In: STOC. (1993) 91–100
3. Saks, M.E., Zaharoglou, F.: Wait-free k-set agreement is impossible: The topology of public knowledge. SIAM J. Comput. **29**(5) (2000) 1449–1483
4. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2) (1985) 374–382
5. Chaudhuri, S.: Agreement is harder than consensus: set consensus problems in totally asynchronous systems. In: PODC. (1990) 311–324
6. Borowsky, E., Gafni, E., Lynch, N.A., Rajsbaum, S.: The BG distributed simulation algorithm. Distributed Computing **14**(3) (2001) 127–146
7. Herlihy, M., Rajsbaum, S.: The decidability of distributed decision tasks (extended abstract). In: STOC. (1997) 589–598
8. Junqueira, F.P., Marzullo, K.: Designing algorithms for dependent process failures. In: Future Directions in Distributed Computing. (2003) 24–28
9. Fitzi, M., Maurer, U.M.: Efficient byzantine agreement secure against general adversaries. In: DISC. (1998) 134–148
10. 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. (2008) 115–124
11. Zielinski, P.: Anti-Omega: the weakest failure detector for set agreement. In: PODC. (2008) 55–64
12. 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) (2004) 333–357
13. Chandra, T.D., Hadzilacos, V., Toueg, S.: The weakest failure detector for solving consensus. In: PODC. (1992) 147–158
14. Lo, W.K., Hadzilacos, V.: Using failure detectors to solve consensus in asynchronous shared-memory systems (extended abstract). In: WDAG. (1994) 280–295
15. Vitanyi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: SFCS, Washington, DC, USA, IEEE Computer Society (1986) 233–243