

Condition-based consensus solvability: a hierarchy of conditions and efficient protocols^{*}

Achour Mostéfaoui¹, Sergio Rajsbaum², Michel Raynal¹, Matthieu Roy¹

¹ IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France (e-mail: {mostefao, raynal, mroy}@irisa.fr)

² Instituto de Matematicas, UNAM, D. F. 04510, Mexico (e-mail: rajsbaum@math.unam.mx)

Received: November 2001 / Accepted: April 2003

Published online: February 6, 2004 – © Springer-Verlag 2004

Abstract. The *condition-based* approach for consensus solvability consists of identifying sets of input vectors, called *conditions*, for which there exists an asynchronous protocol solving consensus despite the occurrence of up to f process crashes. This paper investigates \mathcal{C}_f , the largest set of conditions which allow us to solve the consensus problem in an asynchronous shared memory system.

The first part of the paper shows that \mathcal{C}_f is made up of a hierarchy of classes of conditions, $\mathcal{C}_f^{[d]}$ where d is a parameter (called *degree* of the condition), starting with $d = \min(n - f, f)$ and ending with $d = 0$, where $\mathcal{C}_f^{[0]} = \mathcal{C}_f$. We prove that each one is strictly contained in the previous one: $\mathcal{C}_f^{[d]} \subset \mathcal{C}_f^{[d-1]}$. Various properties of the hierarchy are also derived. It is shown that a class can be characterized in two equivalent but complementary ways: one is convenient for designing protocols while the other is for analyzing the class properties. The paper also defines a linear family of conditions that can be used to derive many specific conditions. In particular, for each d , two natural conditions are presented.

The second part of the paper is devoted to the design of efficient condition-based protocols. A generic condition-based protocol is presented. This protocol can be instantiated with any condition C , $C \in \mathcal{C}_f^{[d]}$, and requires at most $(2n + 1) \lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$ shared memory read/write operations per process in the synchronization part of the protocol. Thus, the value $(f - d)$ represents the “*difficulty*” of the class $\mathcal{C}_f^{[d]}$. An improvement of the protocol for the conditions in $\mathcal{C}_f^{[0]}$ is also presented.

Keywords: Asynchronous system – Collect – Condition – Consensus – Fault-tolerance – Input vector – Snapshot – Shared memory – Step complexity

1 Introduction

The *Consensus* problem lies at the heart of many distributed computing problems one has to solve when designing reliable applications on top of unreliable distributed asynchronous systems. There is a large literature dedicated to studying theoretical and practical aspects of this problem (e.g., [5,21]), that can be informally stated in terms of three requirements. Each process proposes a value, and has to decide on a value (decision and termination) such that there is a single decided value (agreement), and the decided value is a proposed value (validity). One of the most fundamental impossibility results in distributed computing says that this apparently simple problem has actually no deterministic solution in an asynchronous system even if only one process may crash [15]. To circumvent this impossibility, known as FLP, two main approaches have been investigated. One of them consists of relaxing the requirements of the problem, by either allowing probabilistic solutions (e.g., [6]), or approximate solutions (ϵ -agreement [14], or k -set agreement [12]). Another approach consists of enriching the system with synchrony assumptions until they allow the problem to be solved [13]. This approach has been abstracted in the notion of unreliable failure detectors [11]. There have also been studies of hybrid approaches, like combining failure detection with randomization [2,26], or more general approaches for designing algorithms in situations where there is some information about the typical conditions that are encountered when the respective problem is solved [7].

We have recently investigated a *condition-based approach* to tackle the consensus problem [22]. This approach focuses on sets of input vectors that allow n processes to solve the consensus problem despite up to f process crashes, in a standard asynchronous shared memory model. Let an *input vector* be a size n vector, whose i -th entry contains the value proposed by a process p_i . A *condition* (which involves the parameters f and n) is a set of such vectors that can be proposed under normal operating conditions. We are interested in f -fault tolerant protocols that (1) solve consensus at least when such a condition holds, and (2) are always safe. A condition-based consensus protocol solves consensus whenever the input vector belongs to the set of allowed input vectors. Being safe means that the protocol guarantees agreement and validity, whether the proposed input vector is allowed by the condition or not, i.e. we

^{*} Parts of it have previously appeared in [23] and [25].

Correspondence to: Matthieu Roy

only relax the termination property. In addition, we would like the protocols to terminate in well-behaved scenarios even if the input vector does not belong to the condition (for example, they should terminate in all failure-free executions). This is the best we can hope for, since the FLP impossibility result says we cannot require that a consensus protocol terminates always, for every input vector. But, by guaranteeing that safety is never violated, the hope is that such a protocol should be useful in applications. For example, consider the condition “more than a majority of the processes propose the same value.” It is not hard to see that consensus can be solved when the inputs satisfy this condition, when $f = 1$. It is plausible to imagine an application in some real system, that satisfies this condition most of the time; only when something goes wrong, the processes’ proposals get evenly divided, and only then should the protocol take longer to terminate (or even not terminate).

Our previous work [22] on the condition-based approach presented two results. The first is a characterization of the conditions that admit a consensus protocol with the above properties. That is, we described a set of conditions, denoted here \mathcal{C}_f , and proved that there is a consensus protocol for a condition C if and only if $C \in \mathcal{C}_f$. We presented two equivalent combinatorial descriptions of the class \mathcal{C}_f , and described two natural conditions $C1_f$ and $C2_f$ in \mathcal{C}_f that might be useful in practice, and proved them to be maximal (they cannot be extended by the addition of new vectors). The class \mathcal{C}_f is quite rich, since it includes every condition for which there exists a condition-based consensus protocol. The second result is a condition-based protocol that can be instantiated for any particular condition $C \in \mathcal{C}_f$. This protocol has the same step complexity, whatever the particular condition it is instantiated with, namely $O(n \log_2 n)$ read/write shared memory operations in the synchronization part of each process.

Content of the paper. This paper studies the condition-based approach from two perspectives. The first is an investigation of the structure of \mathcal{C}_f (the largest set of conditions), and proposes a systematic way to define conditions. More precisely, the first part, which consists of Sect. 3-4, is made up of the following contributions.

- Although a priori it could be that all conditions of \mathcal{C}_f are equally difficult to solve, it seems plausible that some conditions of \mathcal{C}_f are more difficult to solve than others. If this is the case, there would be more efficient protocols, specially tailored for particular classes of conditions. In practice, one would be interested in identifying the simplest classes of conditions whose input vectors occur frequently, because such classes would perhaps have very efficient consensus protocols. This paper shows that this is indeed the case: the class \mathcal{C}_f is actually made up of a hierarchy of classes of conditions, each one of some *degree* d , starting with $d = \min(n - f, f)$, and ending with $d = 0$. We prove that each one is strictly contained in the next one:

$$\mathcal{C}_f^{\{\min(n-f, f)\}} \subset \dots \subset \mathcal{C}_f^{[d]} \subset \mathcal{C}_f^{[d-1]} \subset \dots \subset \mathcal{C}_f^{[0]} \equiv \mathcal{C}_f.$$

Various structural properties of the hierarchy are studied. Two equivalent characterizations of the classes $\mathcal{C}_f^{[d]}$ are described. The first, *acceptability*, is in terms of the properties that a predicate P and a function S have to satisfy in

order for a protocol to solve consensus for a condition C in $\mathcal{C}_f^{[d]}$; the predicate P tells a process if it can decide based on its view of the input vector, and the function S computes the value to be decided. The second characterization, called *legality*, is in terms of a graph derived from C , d and f . The acceptability characterization is useful for deriving consensus protocols, while the legality characterization is more appropriate to derive noteworthy properties of the hierarchy of conditions such as the following ones: (1) It is decidable in polynomial time if a condition C belongs to a class $\mathcal{C}_f^{[d]}$; (2) If f is too large (namely, $f \geq n - d$), then every condition of degree d is trivial (i.e., protocol will return either a predefined value a , or will not terminate)

- For every degree d , two conditions, denoted $C1_f^{[d]}$ and $C2_f^{[d]}$, are presented. They provide examples of natural conditions that might be useful in practice, and show that the class containments stated above are strict. They generalize the natural conditions $C1_f$, $C2_f$ that were introduced in [22].

A generic form to express a family of conditions is presented. This form is based on a linear combination of weights associated with the values proposed by processes. Whatever be the weight function used to instantiate the generic form, we get, for any pair (f, d) , a condition that belongs to $\mathcal{C}_f^{[d]}$. Interestingly, and in addition to its simplicity, this generic formulation includes the two previous particular conditions. It captures a meaningful set of practically relevant conditions.

The second part of the paper, which consists of Sect. 6-7, concerns the design of condition-based protocols for the shared memory model. It makes the following contributions:

- First, given f, n, d , $0 \leq d \leq f < n$ and a condition $C \in \mathcal{C}_f^{[d]}$, a consensus protocol for C is presented. The correct processes decide and terminate when the actual input vector belongs to the C and there are at most f crashes. It also terminates in other situations, like when a process decides, or when there is no crash. This protocol consists of three parts. The first part allows a process to get an initial view of the input vector, namely, a vector with at least $(n - f)$ input values. The second part, based on an idea presented in [4], synchronizes processes’ views, and executes a number of read/write operations that depends on the degree d of the condition. During each iteration, a process tries to enrich its view of the input vector, in such a way that the views finally obtained by the processes satisfy a containment property: among the final views, any two views that (together) have more than $(f + d)$ undefined entries are ordered. Basically, this means that the degree d of a condition defines the view coherence level needed for the processes to decide consistently: as we shall see, d is a parameter that allow some processes that have enough information to avoid unnecessary synchronization. Finally, the last part of the protocol ensures that the processes terminate in the various scenarios mentioned above.

The complexity of a condition is evaluated in terms of the number of steps of the synchronization part of the protocol. This is motivated by the fact that the other two parts of the protocol are independent of the condition. It is shown that the value $(f - d)$ determines an upper bound on the

“difficulty” of the class $\mathcal{C}_f^{[d]}$ in the sense that the number of read/write operations that are executed by a process in its synchronization, condition-dependent part, is related to $(f - d)$. More precisely, for any condition $C \in \mathcal{C}_f^{[d]}$, this number is proportional to $n \log_2(f - d + 1)$ ¹. Hence, when we progress down in the hierarchy, starting from $d = 0$ with the largest class $\mathcal{C}_f^{[0]}$ by increasing the value of d , there are increasingly efficient protocols for conditions C in $\mathcal{C}_f^{[d]}$, i.e. a condition in $\mathcal{C}_f^{[d+1]}$ admits a more efficient protocol than a condition in $\mathcal{C}_f^{[d]}$.²

- The previous protocol is actually a meta-protocol to be instantiated with a given condition C and values for f, d . Let $\mathcal{P}^{[0]}$ be its instantiation for the conditions $C \in \mathcal{C}_f^{[0]}$. $\mathcal{P}^{[0]}$ considers the largest set of conditions, and requires that $O(n \log_2(f + 1))$ read/write shared memory operations be executed by each process in its synchronization part. This cost is related only to the degree value $d = 0$ (i.e., to the class $\mathcal{C}_f^{[0]}$); more explicitly, this cost is not related to the actual input vector. This means that $\mathcal{P}^{[0]}$ requires $O(n \log_2(f + 1))$ read/write operations even if the input vector actually belongs to a stronger condition $C' \subset C$ with $C' \in \mathcal{C}_f^{[d]}$ and $d > 0$. So, an interesting question is the following: “Considering the class of conditions $\mathcal{C}_f^{[0]}$, is it possible to benefit from the hierarchy to design a protocol \mathcal{P}' whose cost can be less than $O(n \log_2(f + 1))$ when the input vector—which is assumed to belong to a condition $C \in \mathcal{C}_f^{[0]}$ —does actually belong to a stronger condition C' , i.e., such that $C' \in \mathcal{C}_f^{[d]}$ ($d > 0$)?” The paper presents such an adaptive protocol. Its actual cost during an execution depends on several parameters including the input vector, the actual number of failures and the way they are perceived by the processes; its cost is bounded above by $O(n \log_2(f + 1))$.

Theoretical basis and related work. To our knowledge, the idea to consider restricted sets of input vectors for consensus was stated for the first time in [27,28], where possibility and impossibility results in a shared memory system are presented. Moreover, these papers define a strict hierarchy of problems that can be solved in the presence of up to f failures but not in the presence of $(f + 1)$ failures; they illustrate the hierarchy with variants of the consensus problem.

The foundation underlying the proposed condition based approach can be formalized using topology (e.g., [18]). Our setting is not exactly that of the previous topology papers, because those consider *decision tasks* where processes have to always decide, with an output vector satisfying the task specification. Our notion of the problem is a kind of “*safe task*” where, in addition to the requirements of a decision task, processors are required to satisfy a safety property when inputs

¹ For $d \geq f$, the step complexity of the synchronization part is zero since synchronization is no longer needed.

² Following the publication of [23] we discovered a consensus protocol whose cost is $O(n)$ for any $C \in \mathcal{C}_f^{[0]}$ whenever $f < n/2$; see [24]. Thus, this improves on the consensus protocol described here, and originally in [23].

are illegal, without necessarily terminating. From this point of view, our paper is a complexity study of the class of safe tasks with a particular kind of output vectors: all decisions in an execution are equal. In general, the study of f -fault tolerant decision tasks requires higher dimensional topology (except for the case of $f = 1$ which uses only graphs [9]), and leads to undecidable characterizations [16,17] (NP-Hard for $f = 1$ [10]). Due to the simplicity of the allowed output vectors, we are able to derive our results using only graph connectivity. (The main innovation in this context is that the definition of our input graphs depends on the degree d , and it is related to the complexity of the corresponding algorithms.)

Our work might be a first step in the direction of showing interesting lower bounds on the number of read/write operations needed to implement an atomic snapshot operation. The problem of defining and implementing a linearizable snapshot object from single-writer multi-reader registers has been studied since [1]. In the wait-free snapshot protocols presented in [1], each update/snapshot operation requires $O(n^2)$ read and write operations on atomic registers. The best known wait-free simulation of snapshots from read/write operations has $O(n \log n)$ step complexity [4]; hence, our snapshot-based protocol inherits this complexity³. Let us remark that the set of acceptable conditions is quite rich: a condition is not required to satisfy the closure properties needed for the BG-simulation [8] that would allow us to derive results from one level of resilience to another.

Organization of the paper. The paper is made up of eight sections. Section 2 first introduces the computation model and then presents the condition-based approach. Section 3 defines classes of conditions. Section 4 studies the two particular conditions $\mathcal{C}1_f^{[d]}$ and $\mathcal{C}2_f^{[d]}$, and the generic weight-based formula to define conditions. Section 5 presents the hierarchy of classes of conditions. Then, Sect. 6 presents the general condition-based protocol; Sect. 7 makes it adaptive for conditions $C \in \mathcal{C}_f^{[0]}$. Finally, Sect. 8 concludes the paper. Long proofs have been grouped together in the Appendix A.

2 Computation model and the condition-based approach for consensus solvability

2.1 Computation model

We consider a standard asynchronous shared memory system with n , $n > 1$, processes, where at most f , $0 \leq f < n$, processes can crash. The shared memory consists of single-writer, multi-reader atomic registers. For details of this model see any standard textbook such as [5,21].

The shared memory is organized into arrays. The j -th entry of an array $X[1..n]$ can be read by any process p_i with an operation $\text{read}(X[j])$. Only p_i can write to the i -th component, $X[i]$; it uses the operation $\text{write}(v, X[i])$ to write v .

³ If there turns out to exist a linear time implementation of the snapshot operation in single-writer multi-reader register systems, the hierarchy introduced in this paper would collapse. So, the model considered here differs from the shared memory models considered in [3,19,20].

In addition to the shared memory, each process has a local memory. The subindex i is used to denote p_i 's local variables.

To simplify the notation we also consider the following non-primitive, non-atomic **collect** operation which can be invoked by any process p_i . It can only be applied to a whole array $X[1..n]$, and is an abbreviation for $\forall j : \mathbf{do\ read}(X[j]) \mathbf{enddo}$. Hence, it returns an array of values $[a_1, \dots, a_n]$ such that a_j is the value returned by $\mathbf{read}(X[j])$.

2.2 The condition-based approach for consensus solvability

In the *consensus* problem there is a set \mathcal{V} ($|\mathcal{V}| \geq 2$) of values that can be *proposed* by the processes. In an execution, every correct process p_i proposes a value $v_i \in \mathcal{V}$ and all correct processes have to *decide* on the same value v , that has to be one of the proposed values. The proposed values in an execution are represented as an *input vector*, such that the i -th entry contains the value proposed by p_i , or \perp if p_i did not take any step in the execution. We usually use I to denote an input vector with all entries in \mathcal{V} , and J to denote an input vector that may have some entries equal to \perp . As at most f processes can crash, we consider only input vectors J with at most f entries equal to \perp ; these are called *views*. Let \mathcal{V}^n be the set of all possible input vectors with all entries in \mathcal{V} . For $I \in \mathcal{V}^n$, let \mathcal{I}_f be the set of possible views, i.e., the set of all input vectors J with at most f entries equal to \perp , such that I agrees with J in all the non- \perp entries of J . For a set C , $C \subseteq \mathcal{V}^n$, let \mathcal{C}_f be the union of the \mathcal{I}_f 's over all $I \in C$. Thus, in the consensus problem, every vector $J \in \mathcal{V}_f^n$ is a possible input vector.

The *condition-based* approach considers subsets C of \mathcal{V}^n , called *conditions*, that represent common input vectors in a particular distributed application. We are interested in conditions C that, when satisfied (i.e., when the proposed input vector does belong to \mathcal{C}_f), make the consensus problem solvable, despite up to f process crashes. More precisely, we say that a *protocol solves the consensus problem for a condition C and f* if in every execution whose input vector J belongs to \mathcal{V}_f^n , the protocol satisfies the following properties [22]:

- **Validity:** A decided value is a proposed value.
- **Agreement:** No two processes decide different values.
- **Termination:** If (1) $J \in \mathcal{C}_f$ and no more than f processes crash, or (2) all processes are correct, or (3) a process decides, then every correct process decides.

The first two are the usual validity and agreement consensus requirements.

3 Classes of conditions

This section defines and investigates classes of conditions, $(\mathcal{C}_f^{[d]})_{0 \leq d \leq f}$, that allow solving the consensus problem. As we shall see, these classes define a hierarchy. As previously noted, it was proved in [22] that the largest class in the hierarchy, $\mathcal{C}_f = \mathcal{C}_f^{[0]}$, includes every condition for which a consensus protocol does exist. We study the hierarchy from two perspective, namely, acceptability and legality of a condition. These notions were introduced in [22] without the *degree* notion. We define the degree d as the maximum distance beyond f that is

allowed between processes' views of the input vector before these processes need additional synchronization. Intuitively, d can be seen as the maximum disagreement allowed between processes' views of an input vector. Here we generalized the hierarchy to any degree d , $0 \leq d \leq f$.

We use the following notation:

- $\forall J1, J2 \in \mathcal{V}_f^n$, $J1 \leq J2$ if $\forall k : J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$, and we say that $J2$ *contains* $J1$.
- $\#_x(J)$ denotes the number of entries of J whose value is x , with $x \in \mathcal{V} \cup \{\perp\}$.
- For $I \in \mathcal{V}^n$, and $(J_1, \dots, J_\ell) \in \mathcal{I}_f$, $\text{lub}(\{J_1, \dots, J_\ell\})$ is equal to J_0 , the *least upper bound* of the set $\{J_1, \dots, J_\ell\}$ with respect to the \leq operation defined on views.

This least upper bound is well defined (I is an upper bound for this set). It can be obtained from I by replacing an entry with \perp when this entry is \perp in every J_k , i.e., $J_0[k] = \perp$ if $\forall j \in [1..l] : J_j[k] = \perp$, and $J_0[k] = I[k]$ otherwise.

When clear from the context, we sometimes omit mentioning f or d .

3.1 Acceptability and legality

3.1.1 Acceptability

Given a condition C and a value of f , *acceptability* is a combinatorial property of C , inspired by an operational notion defined in terms of a predicate P and a function S : P is a boolean valued function on views of \mathcal{V}_f^n and S is a function from \mathcal{V}_f^n to \mathcal{V} . These functions have to satisfy three properties (related to termination, validity and agreement) in order to design a protocol.

The intuition for the first property is the following. The predicate P allows a process p_i to test whether a decision value can be computed from its view. Thus, P has to return true at least for all views J such that $J \in \mathcal{I}_f$ for $I \in C$.

- Property $\text{T}_{C \rightarrow P}$:
 $I \in C \Rightarrow \forall J \in \mathcal{I}_f : P(J)$.

The second property is related to validity.

- Property $\text{V}_{P \rightarrow S}$:
 $\forall I \in \mathcal{V}^n : \forall J \in \mathcal{I}_f : P(J) \Rightarrow S(J) = \text{a non-}\perp \text{ value of } J$.

The last property concerns agreement. Given an input vector I , if two processes p_i and p_j get the views $J1$ and $J2$, and both belong to \mathcal{I}_f such that $P(J1)$ and $P(J2)$ are satisfied, these processes have to decide the same value of \mathcal{V} , from $J1$ for p_i and $J2$ for p_j , whenever the following holds where the parameter d is called the *degree* of the condition (recall that d represents the maximum disagreement allowed (beyond f) between processes' views before these views get ordered).

- Property $\text{A}_{P \rightarrow S}^{[d]}$:
 $\forall I \in \mathcal{V}^n : \forall J1, J2 \in \mathcal{I}_f : Q1 \wedge Q2 \Rightarrow S(J1) = S(J2)$,
where $\begin{cases} Q1 \equiv P(J1) \wedge P(J2), \\ Q2 \equiv (J1 \leq J2) \vee (\#_{\perp}(J1) + \#_{\perp}(J2) \leq f + d). \end{cases}$

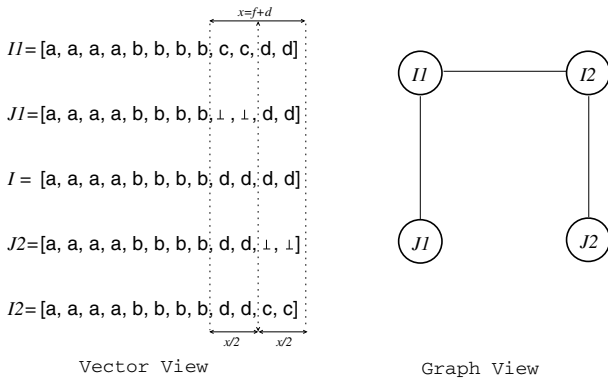


Fig. 1. Case of an edge $(I1, I2)$

Definition 1 A condition C is (f, d) -acceptable if there exist a predicate P and a function S satisfying properties $T_{C \rightarrow P}$, $A_{P \rightarrow S}^{[d]}$ and $V_{P \rightarrow S}$ for f .

3.1.2 Legality

Alternatively, a class of conditions can be analyzed using a representation of its conditions a graph. Given a condition C and a value of f , we associate with it a graph $G^{[d]}(C, f)$ defined as follows. Its vertices are the input vectors I of C plus all their views (i.e., all views J such that $\exists I \in C : J \in \mathcal{I}_f$). Two vertices $I1, I2$ of C are connected by an edge if their Hamming distance $dist(I1, I2) \leq (f + d)$, and two views $J1, J2 \in \mathcal{I}_f$ are connected by an edge if $J1 \leq J2$ (notice that, any $I \in C$ is connected to its views $J \in \mathcal{I}_f$).

Definition 2 A condition C is (f, d) -legal if, for every connected component of $G^{[d]}(C, f)$, there is an input value v that appears in every one of its vertices. □*Theorem 1*

Theorem 1 A condition C is (f, d) -acceptable iff it is (f, d) -legal.

Proof.

\Rightarrow direction: Let C be an (f, d) -acceptable condition with parameters P and S , and consider the graph $G^{[d]}(C, f)$. Let C' be the vertices of one of its connected components. Let us show that S is constant on C' by proving that S maps two connected vertices to the same value. We consider two cases representing the two kinds of edges:

- Case of an edge $(I1, I2)$ with $I1, I2 \in C' \cap C$ (see Fig. 1). By definition of $G^{[d]}(C, f)$, let x be such that $dist(I1, I2) = x \leq (f + d)$. Let $J1 \in \mathcal{I}_{1f}$ be the view obtained by replacing the first $\lceil x/2 \rceil$ entries in which $I1$ and $I2$ differ by \perp . Notice that $\lceil x/2 \rceil \leq f$ because $d \leq f$. Thus, $P(J1)$ is true, by Property $T_{C \rightarrow P}$. Let $J2 \in \mathcal{I}_{2f}$ be the view obtained by replacing the last $\lfloor x/2 \rfloor (\leq f)$ entries in which $I1$ and $I2$ differ by \perp . Thus, $P(J2)$ is true, by Property $T_{C \rightarrow P}$. Let $I = \text{lub}(\{J1, J2\})$. Notice that $\#_{\perp}(I) = 0$ and $J1, J2 \in \mathcal{I}_f$, and I is not necessarily in C . Since $J1, J2 \in \mathcal{I}_f$, $P(J1) \wedge P(J2)$ and $\#_{\perp}(J1) + \#_{\perp}(J2) = x \leq (f + d)$, Property $A_{P \rightarrow S}^{[d]}$ implies that $S(J1) = S(J2)$. Since

$J1 \in \mathcal{I}_{1f}$ and $P(J1) \wedge P(I1)$, Property $A_{P \rightarrow S}^{[d]}$ gives $S(I1) = S(J1)$. Similarly, we have $S(I2) = S(J2)$. Consequently, $S(I1) = S(I2)$.

- Case of an edge $(J1, J2)$ with $J1 \leq J2$: by definition of the graph, let $I \in C$ s.t. $J2 \leq I$. Since $J1, J2 \in \mathcal{I}_f$, Property $T_{C \rightarrow P}$ and Property $A_{P \rightarrow S}^{[d]}$ imply that $S(J1) = S(J2)$.

\Leftarrow direction: Assume C is (f, d) -legal. We have to construct P and S to prove that C is (f, d) -acceptable. Let $P(J) \equiv (\exists I \in C : J \in \mathcal{I}_f)$. For each connected component of $G^{[d]}(C, f)$, there is a non- \perp value v that all its vertices have in common. Let S be a function that returns v for every vertex of this connected component. Clearly, Property $T_{C \rightarrow P}$ and Property $V_{P \rightarrow S}$ hold. We now prove that $A_{P \rightarrow S}^{[d]}$ holds. Consider an $I \in \mathcal{V}^n$, and any two $J1, J2 \in \mathcal{I}_f$, such that $P(J1) \wedge P(J2)$ hold. We consider the two cases of the definition of $A_{P \rightarrow S}^{[d]}$:

- Case $J1 \leq J2$.
 $J1$ and $J2$ are connected in $G^{[d]}(C, f)$. Hence, $S(J1) = S(J2)$ by definition of S .
- Case $\#_{\perp}(J1) + \#_{\perp}(J2) \leq (f + d)$.
We need to show that $S(J1) = S(J2)$. Since $P(J1) \wedge P(J2)$ holds, by definition of P , there exist $I1, I2$ in C , with $J1 \in \mathcal{I}_{1f}$ and $J2 \in \mathcal{I}_{2f}$. Thus, $dist(I1, I) = \#_{\perp}(J1)$ and $dist(I, I2) \leq \#_{\perp}(J2)$. Hence $dist(I1, I2) \leq (\#_{\perp}(J1) + \#_{\perp}(J2))$, which is at most $f + d$, since $d \leq f$. That is, $I1, I2$ are vertices of $G^{[d]}(C, f)$ joined by an edge, and hence $S(I1) = S(I2)$. Similarly, $I1, J1$ and $I2, J2$ are vertices of $G^{[d]}(C, f)$ joined by an edge; hence, $S(I1) = S(J1)$ and $S(I2) = S(J2)$. The equality $S(J1) = S(J2)$ follows and the proof is complete.

3.2 Towards a hierarchy

Here we describe condition classes that allow us to solve the consensus problem, and some of its properties.

Definition 3 The class $\mathcal{C}_f^{[d]}$ consists of all the (f, d) -acceptable conditions (or equivalently, by Theorem 1, all the (f, d) -legal conditions).

Notice that if C is (f, d) -acceptable (equivalently by the previous theorem (f, d) -legal), then C is $(f, d-1)$ -acceptable (i.e., $(f, d-1)$ -legal); that is, $\mathcal{C}_f^{[d]} \subseteq \mathcal{C}_f^{[d-1]}$. This is easily seen using either the acceptability representation, since $A_{P \rightarrow S}^{[d]} \Rightarrow A_{P \rightarrow S}^{[d-1]}$, or the legality representation, since $G^{[d]}(C, f)$ is a subgraph of $G^{[d-1]}(C, f)$. Moreover, for any $d \geq n-f$, we get the same graph $G^{[d]}(C, f)$: it has edges between every pair of vertices. Hence, $\mathcal{C}_f^{[d]} = \mathcal{C}_f^{[d+1]}$ when $d \geq n-f$ and conditions in these classes are trivial; see Theorem 4. Since $d \leq f$, the largest value of d of interest is $d = \min(n-f, f)$. The next theorem follows directly from the previous discussion:

Theorem 2

$$\mathcal{C}_f^{[\min(n-f, f)]} \subseteq \dots \subseteq \mathcal{C}_f^{[d]} \subseteq \mathcal{C}_f^{[d-1]} \subseteq \dots \subseteq \mathcal{C}_f^{[0]} \equiv \mathcal{C}_f$$

We shall later see (Sect. 4.4) the containments are strict when $(f + d) < n$. The next theorem proves that, given a condition C , there is a polynomial (in $|C|$ and n) time algorithm for deciding if C is of degree d .

Theorem 3 *For any pair (f, d) , with $d \leq f$, the class $\mathcal{C}_f^{[d]}$ is decidable in polynomial time.*

Proof. To check if a finite condition C is (f, d) -legal, we consider the graph $G^{[d]}(C, f)$, and check that the vertices of each connected component have at least one input value in common. We can consider only vertices with no entry equal to \perp , because if two such vertices are connected in $G^{[d]}(C, f)$, they are connected without passing through vertices with \perp entries. In this case we have to check that the value in common appears at least $f + 1$ times in each vertex I , to guarantee that it appears also in every view $J \in \mathcal{I}_f$, since these views are in the same connected component of I . Thus, the graph can be constructed in polynomial time as follows. First, $|C|$ vertices are generated. Second, the edges are defined by comparing the n entries of each pair of vertices of C , hence they can be constructed in time $O(n |C|^2)$. The connected components of this subgraph of $G^{[d]}(C, f)$, can be identified in time proportional to the number of edges, which is $O(|C|^2)$, using say, BFS. Once a spanning tree of each connected component G_i has been constructed, the intersection of the values appearing in the vertices of a connected component can be computed in polynomial time. One way of doing this is by considering the set X of values that appear $f + 1$ times in the root of the tree, $|X| \leq n$, and then, for every other vertex of the component, checking if each value $x \in X$ appears $f + 1$ times in it; if not, x is removed from X . This procedure takes time $O(n^2 |G_i|)$, where $|G_i|$ is the number of vertices in G_i . $\square_{\text{Theorem 3}}$

It is clear that every class $\mathcal{C}_f^{[d]}$ is non-empty. But some conditions are trivial in the following sense. For example, consider the condition $C = \{I\}$ that contains a single vector I (with all its entries equal to some non- \perp value). C trivially belongs to \mathcal{C}_f^0 , and hence to all other classes. This motivates the following definition:

Definition 4 *A condition C is trivial if for any pair of parameters (P, S) such that C is (f, d) acceptable, $|S(C)| = 1$, where $S(C) = \{S(I) : I \in C\}$.*

The following theorem, a direct consequence of the legality notion, says that the class $\mathcal{C}_f^{[d]}$ is interesting only when $f \geq n - d$.

Theorem 4 *If $n \leq (f + d)$, every condition C in $\mathcal{C}_f^{[d]}$ is trivial.*

Proof. Let us first observe that $G^{[d]}(C, f)$ has a single connected component because for every pair of vertices $I1, I2 \in C$, $\text{dist}(I1, I2) \leq n \leq (f + d)$, and hence $I1, I2$ are joined by an edge. As $G^{[d]}(C, f)$ contains exactly one connected component, and for any (P, S) pair that makes C (f, d) -acceptable, S is constant on a given connected component, we conclude that $|S(C)| = 1$, i.e., C is trivial. $\square_{\text{Theorem 4}}$

Another property that follows from the legality characterization of a condition is that it is possible to trade fault

tolerance for “efficiency” of a condition⁴, e.g., $\mathcal{C}_{2f}^{[0]} \subseteq \mathcal{C}_f^{[f]}$. More generally, this is expressed by the following “trading” theorem:

Theorem 5 $\mathcal{C}_{f+\alpha}^{[d-\alpha]} \subseteq \mathcal{C}_f^{[d]}$, for $0 \leq \alpha \leq d \leq f$.

Proof. Let us consider a condition $C \in \mathcal{C}_{f+\alpha}^{[d-\alpha]}$, and its graph $G^{[d-\alpha]}(C, f + \alpha)$, where vertices $I1, I2$ are joined by an edge if $\text{dist}(I1, I2) \leq f + \alpha + d - \alpha = (f + d)$. Now let us consider the graph $G^{[d]}(C, f)$. It is a subgraph of $G^{[d-\alpha]}(C, f + \alpha)$, since it is based on the same input vectors, which are connected in the same way: $G^{[d]}(C, f)$ is exactly $G^{[d-\alpha]}(C, f + \alpha)$ where all views that contain more than f values equal to \perp are removed. Since $G^{[d-\alpha]}(C, f + \alpha)$ contains a common element in each connected component, so does $G^{[d]}(C, f)$. That is, C is (f, d) -legal, and hence $C \in \mathcal{C}_f^{[d]}$. $\square_{\text{Theorem 5}}$

4 Two conditions and a generic formulation of conditions

Given d and f , this section introduces two natural conditions, denoted $\mathcal{C}_f^{[d]1}$ and $\mathcal{C}_f^{[d]2}$. These conditions generalize those introduced in [22] that correspond to the case $d = 0$. It then presents a generic weight-based formulation to generate conditions. All the conditions $\mathcal{C}_f^{[d]}$ derived from this formulation are (f, d) -acceptable, i.e., belong to $\mathcal{C}_f^{[d]}$. It is also shown that $\mathcal{C}_f^{[d]1}$ and $\mathcal{C}_f^{[d]2}$ are particular instances of the generic weight-based formulation.

4.1 The conditions $\mathcal{C}_f^{[d]1}$ and $\mathcal{C}_f^{[d]2}$

The condition $\mathcal{C}_f^{[d]1}$. The idea of this condition is to guarantee that the most common value in the input vector I can be unambiguously decided by each process. Some notation is required to formally express it: $\#_{1st}(J)$ denotes the number of occurrences of the most common non- \perp value of J ; $\#_{2nd}(J)$ denotes the number of occurrences of the second most common non- \perp value of J (if there is no such value, $\#_{2nd}(J) = 0$). With this notation, $\mathcal{C}_f^{[d]1}$ can be formally expressed as follows:

$$(I \in \mathcal{C}_f^{[d]1}) \text{ iff } [\#_{1st}(I) - \#_{2nd}(I) > (f + d)].$$

The condition $\mathcal{C}_f^{[d]2}$. The idea of this condition is to guarantee that all the processes have the same extremal (largest or smallest) value in their views in order to decide on it. We (arbitrarily) consider the largest value ($\max(J)$ denotes the largest non- \perp value of J). Formally, we have:

$$(I \in \mathcal{C}_f^{[d]2}) \text{ iff } [a = \max(I) \Rightarrow \#_a(I) > (f + d)].$$

⁴ Efficiency refers here to the cost of the underlying consensus protocol.

4.2 Weight-based definition of conditions

Let w be a function from \mathcal{V} to \mathbb{R}^+ . It associates a positive weight $w(a)$ with each value a that can be proposed. In order to avoid introducing cumbersome notation in the following, an input vector is sometimes considered as the set of the values it contains.

The idea of the generic weight-based formulation of a condition $C \in \mathcal{C}_f^{[d]}$ is to provide a simple way to distinguish a value that can be safely decided from a vector I . It is the following:

$$(I \in C) \equiv \exists a \in I : \forall b \in I, b \neq a : \\ \#_a(I) w(a) - \#_b(I) w(b) > (f + d) \max(w(a), w(b)).$$

Notice that a weight function defines conditions, $C_f^{[d]}$, one for each value of d . And from this definition it follows that $\dots \subseteq C_f^{[d+1]} \subseteq C_f^{[d]} \subseteq \dots \subseteq C_f^{[0]}$.

The intuition that underlies this definition is the following. For a value a of an input vector I to be decided (i.e., for I to belong to the condition C), this value has to “bypass” any other value b present in I , (1) despite up to f process crashes, and (2) whatever the number occurrences of that “adversary” value b . This means that:

- The value a has to be “present enough” despite the previous situations. This is expressed by the following constraint: $(\#_a(I) - (f + d)) w(a) > \#_b(I) w(b)$.

- The value a has to be “distant enough” from any b to be distinguishable. This is expressed by the following constraint: $\#_a(I) w(a) > (\#_b(I) + (f + d)) w(b)$.

The weights are used to represent the respective “power” of each value of \mathcal{V} . The generic definition is simply the combination of the two previous constraints involving the weights and the parameter $f + d$.

The theorem that follows (its proof is given in Appendix A.1) states that any condition C defined by the previous generic weight-based expression defines an (f, d) -acceptable condition, i.e., belongs to $\mathcal{C}_f^{[d]}$.

Theorem 6 $\forall d \geq 0$, any function w from \mathcal{V} to \mathbb{R}^+ defines a condition $C \in \mathcal{C}_f^{[d]}$, with the following parameters:

$$P(J) \equiv \exists a \in J : \forall b \in J, b \neq a, \\ (\#_a(J) + \#_{\perp}(J))w(a) - \#_b(J)w(b) \\ > (f + d) \max(w(a), w(b))$$

$$S(J) = a \text{ s.t. } \#_a(J)w(a) = \max\{\#_b(J)w(b) : b \in J\} \\ (\text{the value distinguished by } P(J)).$$

4.3 Examples of conditions

We consider here two weight functions that actually define the conditions $C1_f^{[d]}$ and $C2_f^{[d]}$ previously introduced. They differ in the way they favor values. Other weight functions defining new conditions can easily be defined [25].

Favoring no value. Let us first consider the *uniform weight* function: $\forall a \in \mathcal{V} : w(a) = 1$. This function favors no value. The generic expression simplifies and becomes:

$$(I \in C) \equiv (\exists a \in I : \forall b \in I, b \neq a : \\ \#_a(I) - \#_b(I) > (f + d)).$$

As we can see, this is the condition family $C1_f^{[d]}$ defined in Sect. 4.1. It favors the value that appears the most often in an input vector: to be decided, despite up to f crashes and the presence of any other value b , the most common value a has to appear $(f + d)$ times more than b .

Largely spacing out the favors. Let $\mathcal{V} = \{a_1, \dots, a_p\}$, and let us assume that these values are ranked: $a_1 < a_2 < \dots < a_{p-1} < a_p$. Moreover, let $w(a_i) = n^i$. This condition favors a_p (the largest value in \mathcal{V}). Then, if a_p is not proposed it favors a_{p-1} (the second largest value in \mathcal{V}), etc.

As all weights are powers of n , we can simplify the generic formula. Let a_i be the largest value that appears in a vector I , and a_j another value in I (hence $i > j$). The constraint part in the formula becomes: $\#_{a_i}(I) n^i - \#_{a_j}(I) n^j > (f + d) \max(n^i, n^j)$, which can be simplified into $\#_{a_i}(I) - \#_{a_j}(I) (1/n^{i-j}) > (f + d)$. As $i - j > 1$ and $\#_{a_j}(I) < n$, we have $0 < \#_{a_j}(I) (1/n^{i-j}) < 1$. This allows us to simplify once more and we get:

$$(I \in C) \equiv (a_i = \max(I) \Rightarrow \#_{a_i}(I) > (f + d)).$$

which is the condition family $C2_f^{[d]}$: a vector belongs to this condition if its greatest value appears more than $(f + d)$ times.

4.4 (f, d) -acceptability of $C1$ and $C2$

Lemma 1 $C1_f^{[d]}$ is (f, d) -acceptable with the following parameters:

- $P1_f^{[d]}(J) \equiv \#_{1st}(J) - \#_{2nd}(J) > (f + d - \#_{\perp}(J))$,
- $S1(J) = a$ such that $\#_a(J) = \#_{1st}(J)$.

Proof. The (f, d) -acceptability of $C1_f^{[d]}$ follows from Theorem 6 and the uniform weight function defined in Sect. 4.3. The acceptability parameters $P1_f^{[d]}$ and $S1$ follow from Theorem 6. $\square_{\text{Lemma 1}}$

Lemma 2 $C2_f^{[d]}$ is (f, d) -acceptable with the following parameters:

$$P2_f^{[d]}(J) \equiv a = \max(J) \Rightarrow (\#_a(J) > (f + d - \#_{\perp}(J))), \\ S2(J) = \max(J).$$

Proof. The (f, d) -acceptability of $C2_f^{[d]}$ follows from Theorem 6 and the weight function $w(a_i) = n^i$ defined in Sect. 4.3. The acceptability parameters $P2_f^{[d]}$ and $S2$ follow from Theorem 6. $\square_{\text{Lemma 2}}$

Using $C2_f^{[d]}$, the following lemma proves that there is a (f, d) -legal condition that is not $(f, d + 1)$ -legal, thus showing that condition classes containments are strict with respect to d .

Lemma 3 Assume $(f + d) < n$ and $d < f$. Then, $C2_f^{[d]}$ is not $(f, d + 1)$ -legal.

Proof. Assume for contradiction that $(f + d) < n$ and $C2_f^{[d]}$ is $(f, d + 1)$ -legal. Consider its graph $G^{[d+1]}(f, C2_f^{[d]})$. Consider the following vertices of $C2_f^{[d]}$. $I1$ has the first $(f + d + 1)$ positions equal to a value a , and the others equal to b , such that $a > b$, and $I2$ has all its entries equal to b . Clearly, $I1$ is in the same connected component as the vector with all entries equal to a , since we can switch one by one the b 's of $I1$ to a 's while maintaining the property that the number of a 's is more than $(f + d)$. Thus, there is one, and only one value in common to all vectors of this component, namely, a . On the other hand, $I2$ has only b 's, so it is in a different connected component. However, $\text{dist}(I1, I2) = (f + d + 1)$, and hence there is an edge connecting $I1$ and $I2$ in $G^{[d+1]}(f, C2_f^{[d]})$, a contradiction. $\square_{\text{Lemma 3}}$

The next corollary follows directly from the two previous lemmas.

Corollary 1 Let $(f + d) < n$ and $d < f$. We have $C_f^{[d]} \neq C_f^{[d+1]}$.

5 The hierarchy

The next theorem provides the complete hierarchy of classes of conditions.

Theorem 7

$$C_f^{[\min(n-f, f)]} \subset \dots \subset C_f^{[d]} \subset C_f^{[d-1]} \subset \dots \subset C_f^{[0]} \equiv C_f.$$

Proof. Class containments follow from Theorem 2. The fact that the containments are strict when $(f + d) < n$ is an immediate consequence of Corollary 1. $\square_{\text{Theorem 7}}$

6 A condition-based consensus protocol for $C_f^{[d]}$

This section presents a condition-based protocol (Fig. 2) that solves the consensus problem for any condition $C \in C_f^{[d]}$, once its parameters P and S have been appropriately instantiated. More general than the protocol introduced in [22] (which works only for $d = 0$), it is based on a `strong_collect` procedure that partially orders views. As announced in the Introduction, this protocol is efficient, general (in the sense that it works for any $f < n$), and allows the correct processes to terminate in many cases, despite crashes and the fact that the input vector does not always belong to the condition.

6.1 The consensus protocol

Our consensus protocol uses a `strong_collect` abstraction, that aims at taking into account and exploiting the degree d of a condition. We first define its specification and, based on it, prove the correctness of the consensus protocol. Then, the

next subsection presents an implementation of `strong_collect`. This procedure (including the auxiliary classifier improver called `classifier` in [4]) is a direct generalization of the `scate` procedure of Attiya and Rachman in [4] (in turn inspired by [3]).

The strong_collect abstraction. The goal of this abstraction is to provide processes with views of the proposed values that are ordered by containment when their numbers of \perp values exceed some threshold (namely, $f + d$). Due to Property $A_{P \rightarrow S}^{[d]}$, the views including “few” \perp values are guaranteed to provide a consistent decision (if any). A `snapshot` abstraction (such as [1,4]) could be used instead. But as a `snapshot` orders *all* views by containment, it would be more expensive than the `strong_collect` abstraction which is not required to order all views. As d increases, the step complexity (i.e., the number of atomic read/write operations on shared variables) of the `strong_collect` abstraction decreases, until it becomes a void statement when $d = f$. Conversely, when d decreases, `strong_collect` orders more views. More precisely, to correctly implement the `strong_collect` abstraction, a protocol should satisfy the following specification, where I is the input vector of the execution of the consensus protocol that invokes `strong_collect`:

Specification `strong_collect` $_f^{[d]}$ is an n process wait free protocol. Assume a set of processes $\{p_i\}$ invoke it with views $\{J_i\}$ in \mathcal{I}_f . Let $J = \text{lub}(\{J_i\})$. Then, each process p_i (resp. p_j) eventually gets back a view J'_i (resp. J'_j), such that:

1. $J_i \leq J'_i \leq J$,
2. $(\#_{\perp}(J'_i) + \#_{\perp}(J'_j) > f + d) \Rightarrow (J'_i \leq J'_j \vee J'_j \leq J'_i)$,
3. The number of read/write operations executed by a process p_i is $O(n \log(f - d + 1))$.

The consensus protocol. Let f and d be two integers, with $d \leq f < n$ and $d \leq (n - f)$. The protocol assumes P and S have been instantiated to correspond to a condition C that belongs to the class $C_f^{[d]}$ (i.e., P, S satisfy Property $T_{C \rightarrow P}$, Property $V_{P \rightarrow S}$, and Property $A_{P \rightarrow S}^{[d]}$ for f). A process p_i starts executing the protocol by invoking the function `Consensus` $_f^{[d]}(v_i)$ where v_i is the value it proposes. It terminates when it executes the statement `return` which provides it (at line 6, 8 or 11) with the decided value.

The shared memory is made up of two arrays of shared atomic registers, $V[1..n]$ and $W[1..n]$. Both are initialized to $[\perp, \dots, \perp]$. The local variables of a process p_i are subindexed with i . The protocol has the three-part structure described in the Introduction:

Part 1 The processes build their views (lines 1-2). A process p_i first writes its input value v_i to the shared array V . Then p_i repeatedly reads V until at least $(n - f)$ processes (including itself) have written their input values in V , from which it constructs its initial view J_i , where $J_i[j]$ is the input value of p_j , or \perp if p_j has not yet written its input value.

Part 2 Condition-dependent, synchronization part of the protocol (lines 3-6). Now, p_i uses the `strong_collect` $_f^{[d]}$ underlying abstraction to compute a possibly enriched view J'_i

according to Specification 6.1. With the view J'_i , it checks whether it can make a decision, by evaluating $P(J'_i)$. If true, p_i returns $S(J'_i) = w_i$ (line 6), otherwise (i.e., $w_i = \perp$) p_i proceeds to the last part of the protocol. In either case, it writes first its decision (or \top if it could not decide) in the shared variable $W[i]$ to help other processes decide in the next part.

Part 3 Termination (lines 7-11). In this section, p_i enters a loop to look for a decision value (i.e., a value different from \perp , \top) provided by another process p_j in the shared variable $W[j]$. If, while waiting for a decision, p_i discovers that every process has written a value to W , and no process can directly decide (all these values are \top), p_i concludes that every process has deposited its initial value in the shared array V in line 1. Then, p_i reads V (line 10) to get the full input vector, and proceeds to decide according to a fixed, deterministic rule F that returns one of the input values (such as max).

As indicated in the Introduction, we consider only the synchronization part of the protocol for the step complexity, that is, the shared memory operations performed by the `strong_collect` subprotocol. This is motivated by the fact that the other parts (although they depend on f and the actual number of failures) are independent of the condition, namely, d and C .

Theorem 8 *The Consensus $_f^{[d]}$ protocol solves the consensus problem for f and any condition C if its parameters P, S satisfy Property $T_{C \rightarrow P}$, Property $V_{P \rightarrow S}$, and Property $A_{P \rightarrow S}^{[d]}$. The step complexity of a process is $O(n \log(f - d + 1))$.*

Proof. Validity: It follows from the code, from Property $V_{P \rightarrow S}$, and from the first item of Specification 6.1, that a decided value is a proposed value.

Agreement: To prove that no two processes decide different values, let us first consider two processes, p_i and p_j that decide in line 6. Thus, $P(J'_i)$ and $P(J'_j)$ are true, and they decide $S(J'_i)$ and $S(J'_j)$, respectively. If $\#_{\perp}(J'_i) + \#_{\perp}(J'_j) \leq f + d$, then we get $S(J'_i) = S(J'_j)$ from the second part of Property $A_{P \rightarrow S}^{[d]}$. If $\#_{\perp}(J'_i) + \#_{\perp}(J'_j) > f + d$, then, due to second item of Specification 6.1, $J'_i \leq J'_j$ (or the opposite), and consequently we get $S(J'_i) = S(J'_j)$ from the first part of Property $A_{P \rightarrow S}^{[d]}$.

Now, let us assume that p_i decides at line 8. Then, it decides a value decided by another process at line 6, and we are done. Finally, assume p_i decides in line 11. Then p_i gets $X_i[k] = \top$ for all k , and hence no process decides at lines 6 or 8. Moreover, all processes that decide get the complete input vector, and as they all apply the same (deterministic) function F to this vector, they get the same decided value.

Termination: There are three cases of termination: (1) when the input vector belongs to the condition, (2) when all processes are correct, and (3) when a correct process decides.

- Suppose that the input vector is within the condition, i.e. $I \in C$. Let p_i be a correct process. As there are at least $n - f$ correct processes, p_i does not block forever at line 2, and consequently p_i gets a view V_i . As $I \in C$, it follows from $T_{C \rightarrow P}$ that $P(V_i)$ is true. From $V_{P \rightarrow S}$ and lines 3-4, we get that $w_i \neq \perp, \top$. Hence, at line 6, p_i decides and terminates.

- If all processes are correct, then every process will eventually write a non- \perp value in W at line 5. Hence, the `repeat` loop at line 7-9 will eventually end for every process, allowing processes to terminate at least at line 11.
- Let p_i be a process that decides and terminates.
 - If p_i decides at line 11, then every process is correct in the execution, and termination is ensured using the previous item of the proof.
 - If p_i terminates at line 6, it has written its decision value into $W[i]$ at line 5. Hence, every correct process will eventually read $W[i]$ at line 8 and terminate.
 - If p_i terminates at line 8, it decides a value from W , say $W[k]$. This value was previously written by p_k at line 5; termination is ensured using the above case.

Step Complexity: Follows directly from item 3 of Specification 6.1. \square Theorem 8

Remark. The previous theorem shows that the consensus protocol described in Fig. 2 satisfies the **Termination** property, namely, the correct processes terminate when (1) the actual input vector J is such that $J \leq I$ for $I \in C$ and no more than f processes crash, or (2) all processes are correct, or (3) a process decides. The reader can easily see that the protocol allows the correct processes to terminate in more cases, namely when all processes execute line 5, or when the view of a process p_i that has executed line 5 has an extension in C .

6.2 Adapting Attiya-Rachman's synchronization tree

This section describes the implementation of the `strong_collect` abstraction (see Fig. 4). Basically, an execution of `strong_collect` by a process p_i traverses a labeled binary tree T , starting at the root with its initial view J_i , and then going down one level in each iteration of its loop (lines 4-5), until it terminates in a leaf with a final view J'_i . At each vertex of the tree, processes propose views, which get refined into two categories: “rich” views proceed to the right child of the vertex, and the other views proceed to the left child. The aim of a tree traversal is to allow processes to classify and enrich their views, and ensure that the final views obtained by processes that have “too many” \perp entries (those views are at the left of the tree) are dominated by the views that have fewer \perp entries (those views are at the right of the tree).

Notice that f and d do not appear in any line of the protocol. These parameters affect only the depth of the tree T and its labeling. So, we first describe the behavior of the protocol for arbitrary trees. We will then show that there exists a particular tree for which the proposed protocol satisfies the `strong_collect` requirements of Specification 6.1.

Traversing a labeled binary tree.

The tree used by `strong_collect` is an adaptation of the synchronization tree proposed by Attiya and Rachman [4]. It consists of a static data structure shared by the processes that can access it from the pointer *root*. Each non-leaf vertex v contains (1) pointers to its children (*v.left* and *v.right*), and (2) an array *v.R* of n vectors. The vector *v.R*[i] is a shared variable

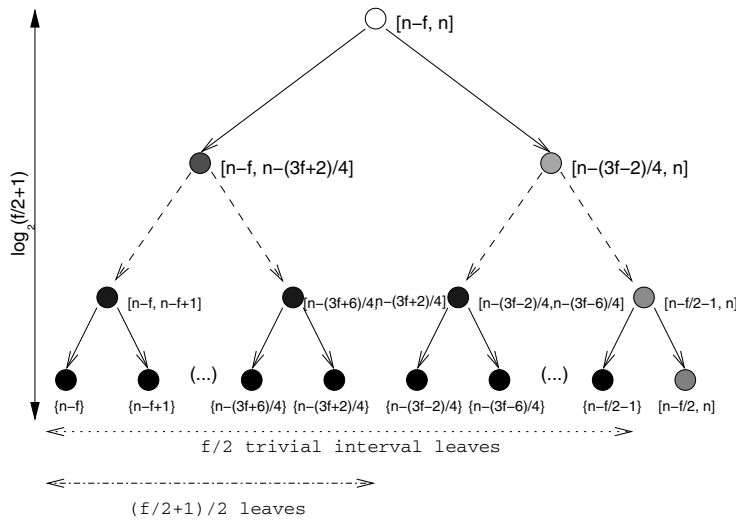


Fig. 3. A tree (with $d = 0$) adapted from Attiya-Rachmann's tree

Designing an appropriate tree.

We now design a tree $T_f^{[d]}$ ($0 \leq d \leq f$) to be used by the protocol $\text{strong_collect}_f^{[d]}$. We assume $d < f$, because otherwise the protocol is not needed: assume $d \geq f$, since every view contains at most $f \perp$ entries, two views cannot have more than $2f \leq f+d$ undefined entries, thus do not need to be ordered by containment. To satisfy Specification 6.1(2), we want views J_i, J_j to be ordered whenever $\#_{\perp}(J_i) + \#_{\perp}(J_j) > f + d$; that is, whenever $|J_i| + |J_j| < 2n - (f + d)$. Thus, we want the tree to have a leaf v with interval $L(v) = H(v) = x$ for every $n - f \leq x < \lceil (2n - (f + d))/2 \rceil$. As we shall see, these are the only leaves that require trivial intervals (i.e., of size 1); we use one more leaf with interval $[\lceil (2n - (f + d))/2 \rceil, n]$. This leaf can have a non-trivial interval, since we do not need the corresponding views to be ordered. Using these intervals for the leaves, the interval of every other vertex is defined inductively, as the union of the intervals of its children. For this to work, we assume the number of leaves, $\lceil (2n - (f + d))/2 \rceil - (n - f) + 1 = \lceil (f - d)/2 \rceil + 1$, to be a power of two (standard techniques can be used otherwise). It follows that, in the general case, the tree $T_f^{[d]}$ has a depth equal to $\lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$. We can now prove that $\text{strong_collect}_f^{[d]}$ satisfies its specification when using the tree $T_f^{[d]}$.

The tree depicted in Fig. 3 actually corresponds to the case $f = 2^p - 2$ (for some positive integer p) and $d = 0$.

Lemma 5 *The $\text{strong_collect}_f^{[d]}$ protocol described in Fig. 4 with tree $T_f^{[d]}$ satisfies Specification 6.1.*

Proof. The proof of Specification 6.1(1) is a direct consequence of the `classifier_improver` Specification 6.2(1).

To prove Specification 6.1(2), we consider J'_i, J'_j (two returned views) with $\#_{\perp}(J'_i) + \#_{\perp}(J'_j) > f + d$; that is, with $|J'_i| + |J'_j| < 2n - (f + d)$. Let us first consider the case where $|J'_i| < \lceil (2n - (f + d))/2 \rceil$ and $|J'_j| < \lceil (2n - (f + d))/2 \rceil$. Thus, both views end in leaves with trivial intervals, and either one is less than the other, by Lemma 4(3) or else they are equal by Lemma 4(4). The other case is when $|J'_i| <$

$\lceil (2n - (f + d))/2 \rceil$ and $|J'_j| \geq \lceil (2n - (f + d))/2 \rceil$ (or the opposite). Then, $J'_i < J'_j$ by Lemma 4(3).

The proof of Specification 6.1(3) follows from (1) the height of $T_f^{[d]}$ which is $\lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$, and (2) the fact that a process executes one `write` and at most two `collect` operations along each vertex on a path from the `root` to a leaf.

□*Lemma 5*

Corollary 2 *The step complexity of the `strong_collect` protocol (i.e., the step complexity of the synchronization part of the `Consensus_f^{[d]}` protocol) is at most*

$$(2n + 1) \lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil .$$

Proof. Immediate consequence of the height of the tree $T_f^{[d]}$, and the fact that a `collect` costs n steps. □*Corollary 2*

6.3 Implementing the `classifier_improver` abstraction

As we have seen, the aim of the `classifier_improver` abstraction associated with the vertex v is to ameliorate the current views of processes p_i visiting this vertex v , orienting them to the left or right subtree rooted at v , according to the size and the content of their current views. A protocol implementing this abstraction for a vertex v is described in Fig. 5. It works as follows. The processes write their views J_i to a shared array $v.R$, and end up with views J'_i of size either at most $H(v.\text{left})$ or greater than $H(v.\text{left})$. Processes also get back a boolean value rich_i that indicates which of the two cases occurred. Hence, this procedure is invoked for each non-leaf vertex v , and each of these invocations is independent of the others.

Lemma 6 *Let us consider a vertex v . The `v.classifier_improver` protocol described in Fig. 5 satisfies Specification 6.2.*

Proof. The proof of the first item of Specification 6.2 follows directly from the code and the fact that, $\forall J_i, \dots, J_j$, we have $J_i \leq \text{lub}(\{J_i, \dots, J_j\})$.

The proof of Specification 6.2(2) follows directly from the lines 3-6 (test of line 3 and the fact that the view J_2 returned

```

Function strong_collectf[d](Ji):
(1) currenti ← Ji;
(2) v ← root;
(3) while v is not a leaf do
(4)     (currenti, richi) ← v.classifier_improver(currenti);
(5)     if richi = yes then v ← v.right
           else v ← v.left endif
(6) endwhile;
(7) return(currenti)  % (J'i = currenti) %

```

Fig. 4. An implementation of strong_collect_f^[d]

by a collect issued after a collect that returned $J1$ satisfies: $J1 \leq J2$.

For the proof of Specification 6.2(3) let us observe that, as $rich_j = yes$, p_j executes two collect operations, which we call collect1 (line 2) and collect2 (line 4). There are two cases.

- If p_i executes its write (line 1) before p_j starts collect2, this collect gets the value written by p_i , and consequently $J_i \leq J'_j$. As $rich_i = no$, it follows that $J'_i = J_i$ (from the protocol code). Moreover, as $rich_i = no$ and $rich_j = yes$, we have $|J'_i| \leq H(v.left) < |J'_j|$ from the test at line 3. Hence, $J'_i \neq J'_j$. It follows that $J'_i < J'_j$.

- Let us now show that the other case does not occur: suppose that p_i executed its write (line 1) after p_j starts collect2. Hence, p_i wrote after the end of collect1 by p_j . It follows that when p_i executes collect (line 2) it gets at least as many values as collect1 (because it started after collect1 terminated). Consequently, p_i sees a collected view of size at least as large as p_j saw in that line, which is greater than $H(v.left)$. It follows that $rich_i = yes$, a contradiction.

The proof of Specification 6.2(4) follows directly from the code (no entry value is created by the protocol).

For the proof of Specification 6.2(5), let us first notice that for every process p_i such that $rich_i = no$, we have $J'_i = J_i$. Now, let us consider the set of processes p_j that get $rich_j = no$. Among them, let p_k be the last that executed line 1. Due to the linearizability property on the basic write and read operations, when p_k executes line 2, it sees the inputs of all the processes p_j such that $rich_j = no$. Thus, the size of the lub of these inputs must be at most $H(v.left)$, since otherwise p_i would get $rich_i = yes$ when it executes line 3. $\square_{Lemma 6}$

7 An adaptive protocol for conditions in \mathcal{C}_f

As noticed in the Introduction, the previous protocol is actually a meta-protocol designed for any pair (f, d) . So, let $\mathcal{P}^{[d]}$ and $\mathcal{P}^{[0]}$ be its instances tailored for the conditions in $\mathcal{C}_f^{[d]}$ and $\mathcal{C}_f^{[0]}$, respectively, and let us consider a condition $C \in \mathcal{C}_f^{[0]} - \mathcal{C}_f^{[d]}$. When the input vector belongs to C , it is possible that $\mathcal{P}^{[d]}$ does not terminate, while $\mathcal{P}^{[0]}$ terminates. So, $\mathcal{P}^{[0]}$ terminates in more circumstances than $\mathcal{P}^{[d]}$. On the other hand, when considering their costs (in terms of read/write shared memory operations per process in their synchronization part), the costs of $\mathcal{P}^{[d]}$ and $\mathcal{P}^{[0]}$ are $O(n \log_2(f-d+1))$ and $O(n \log_2(f+1))$,

```

Function v.classifier_improver(Ji):
(1) write(Ji, v.R[i]);
(2) Ri ← collect(v.R);
(3) if |lub({Ri[1], ..., Ri[n]})| > H(v.left)
(4)     then Ri ← collect(v.R);
(5)     return (lub({Ri[1], ..., Ri[n]}) ,yes)
(6)     else return (Ji ,no)
(7) endif

```

Fig. 5. An implementation of classifier_improver

respectively⁵. Hence, the following question stated in the Introduction: “Considering the class of conditions $\mathcal{C}_f^{[0]}$, is it possible to benefit from the hierarchy to design a protocol \mathcal{P}' whose cost can be less than $O(n \log_2(f+1))$ when the input vector -which is assumed to belong to a condition $C \in \mathcal{C}_f^{[0]}$ - does actually belong to a stronger condition C' , i.e., such that $C' \in \mathcal{C}_f^{[d]}$ ($d > 0$)?” This section presents such an adaptive protocol that, according to the number of current failures and the way they are perceived by the processes, requires that a process executes $O(n \log_2(f+1))$ read/write shared memory operations (in its synchronization part) only in worst case scenarios. So, the interesting point is that the adaptive protocol never costs more than $\mathcal{P}^{[0]}$ and costs much less when the input vector belongs to a condition $C \in \mathcal{C}_f^{[d]}$ with d close to f .

The idea that underlies this adaptive protocol is to allow processes to decide a value during their tree traversal, before reaching a leaf of the tree. What is interesting is that this only slightly increases local computation, while reducing communication between processes. Hence it could be useful for architectures such as non-uniform memory access distributed shared memory (e.g., NUMA DSM), where memory access time might be high and unpredictable. In general, any architecture for which local computation cost is negligible with respect to communication cost would benefit from this protocol.

Let us consider the following example that uses the condition family $\mathcal{C}1$. Suppose that $n = 10$, $f = 3$, and 2 processes propose 0, while 8 processes propose 1. In this execution, every process will eventually decide 1 and terminate in $O(n \log_2(f+1))$ steps. A process that gets the view $J = [1, 1, 1, 1, 1, 1, 1, \perp, 0, \perp]$ after the collect invocation satisfies $P1_f^{[d]}(J)$ for $d \in [0, 4]$. The proposed adaptive protocol

⁵ Notice the cost of $\mathcal{P}^{[0]}$ is $O(n \log_2(f+1))$, which is less than the cost of the protocol presented in [22] that had $O(n \log_2 n)$ step complexity.

will benefit from this, and indeed will permit the process to decide at the root of the tree, without traversing the synchronization tree.

7.1 Structure of the protocol

This protocol is the same as the protocol of Fig. 2 except that `strong_collect` is replaced by a new subprotocol, namely, `Decision_Chasing`. More precisely, the lines 3-4 of Fig. 2 are replaced by:

$$(3)(4) \quad w_i \leftarrow \text{Decision_Chasing}(J_i).$$

The `Decision_Chasing` function is the core of the protocol from safety and efficiency point of views: it returns the decided value if p_i can decide by itself, or \perp if it cannot. This function relies on the traversal of a $T_f^{[d]}$ tree as defined in Sect. 6.2.

As we are interested in the conditions of the class $\mathcal{C}_f^{[0]}$, the tree considered is $T_f^{[0]}$. Such a tree is depicted in Fig. 3. The next part shows how and under what conditions the traversal of this tree can be appropriately shortened (according to the information currently at the process's disposal) in order to reduce shared memory accesses.

7.2 Shortening the tree traversal

The idea that underlies the design of the `Decision_Chasing` function described in Fig. 6 is to allow a process to stop its tree traversal before reaching a leaf. This will obviously decrease the length of the path traversed by the process and consequently allow it to decide early thereby reducing the cost of its synchronization part. The difficulty in attaining this goal consists in ensuring that, whatever the vertices at which two processes p_i and p_j stop their tree traversal, they will decide the same non- \perp value: the consensus agreement property must be guaranteed whatever parts of the tree are traversed by processes. Moreover, given a condition $C \in \mathcal{C}_f^{[0]}$, the protocol must terminate at least when $I \in C$.

Underlying idea of the protocol. Let $C \in \mathcal{C}_f^{[0]}$. As we have seen, C gives rise to a hierarchy of conditions $\dots \subset C^{[d+1]} \subset C^{[d]} \subset \dots \subset C^{[0]} = C$. Let $P^{[d]}$ be a predicate associated with $C^{[d]}$.

We use the following intuitive idea: when d increases, the maximum allowed difference between processes' views increases. Processes enter the tree traversal with arbitrary views, and progress downwards, ordering their views. A process that reaches a leaf checks if $P^{[0]}$ holds, but it may be possible that, in every vertex, there exists some predicate $P^{[d]}$ ($d > 0$) that would have permitted the process to decide early.

When a process p_i visits a vertex v and gets J as current view, it tries to benefit from the fact that J may belong to some \mathcal{I}_f such that $I' \in C^{[d]}$, in order to stop its tree traversal. A process has to traverse the tree from the root to a leaf only in the worst case.

In order for the agreement property to be never violated by processes stopping at arbitrary vertices, the protocol requires that the predicates $P^{[d]}$ satisfy monotonicity properties.

Properties on the sequence of predicates. When it visits a vertex v , a process has first to compute a value d to test whether $P^{[d]}(J)$ holds (J being its current view). The determination of d is crucial for the protocol correctness: d has to be “as small as possible” to allow early tree exit, but large enough to guarantee agreement.

Let us consider two processes that exit the tree at $v1$ and $v2$ with the views $J1$ and $J2$. This means that both $P^{[d1]}(J1)$ and $P^{[d2]}(J2)$ hold. To ensure that we also have $S(J1) = S(J2)$, the protocol requires the following properties be satisfied:

- Decreasing Degree Monotonicity (\mathcal{M}_{dd})

$$\forall d > 0, \forall J \in \mathcal{V}_f^n : P^{[d]}(J) \Rightarrow P^{[d-1]}(J).$$

- Increasing Vector Monotonicity (\mathcal{M}_{iv}) $\forall d \geq 0, \forall J \in \mathcal{V}_f^n, \forall d' \leq d, \forall J' \geq J :$

$$\left(P^{[d]}(J) \wedge \text{dist}(J, J') \leq \frac{d-d'}{\alpha(J)} \right) \Rightarrow P^{[d']}(J').$$

Let J be a view such that $P^{[d]}(J)$ holds. This means that $\exists I \in C^{[d]}$ with $J \in \mathcal{I}_f$. As $C^{[d]} \subset C^{[d-1]}$, it follows that $I \in C^{[d-1]}$. Hence $P^{[d-1]}(J)$ holds, and the sequence of predicates $P^{[d]}$ of any condition family satisfies \mathcal{M}_{dd} .

In contrast to \mathcal{M}_{dd} , the property \mathcal{M}_{iv} involves a parameter α for each J . It follows that for the sequence of predicates $P^{[d]}$ to satisfy \mathcal{M}_{iv} , the value $\alpha(J)$ associated with each view J actually depends on the condition (this issue is addressed in Sect. 7.4). The intuition that underlies the \mathcal{M}_{iv} property is the following. For each J , there is a “disc of augmented vectors” J' , centered at J and with radius $\frac{d-d'}{\alpha(J)}$. The property \mathcal{M}_{iv} requires that all those augmented vectors satisfy $P^{[d']}$. As we start from d and J , and proceed to $d' \leq d$ and $J' \geq J$, \mathcal{M}_{iv} states a tradeoff relating how much an increase in the number of non- \perp values in a view (from J to J') affects a degree reduction (from d to d') when we want to have both $P^{[d]}(J)$ and $P^{[d']}(J')$.

Let p_i and p_j be two processes with current views J and J' , respectively. Let us consider the extreme case where $d' = 0$ and $d = \alpha(J) \text{dist}(J, J')$. If $P^{[d]}(J)$ holds (allowing p_i to decide early), the property \mathcal{M}_{iv} allows p_j to also decide. So, when p_i is visiting vertex v and has J as current view, the fact that it does not know J' means that it has to use some value to consistently approximate $\text{dist}(J, J')$. To that aim, it considers the value $\#_{\perp}(J) + n - L(v) - f$. As d cannot be negative, it actually uses $\max(0, \alpha(J) \times (\#_{\perp}(J) + n - L(v) - f))$ to define the value of d it uses to compute $P^{[d]}(J)$. The proof (see Lemma 7) will show that this heuristic value is consistent.

7.3 The core of the protocol

The `Decision_Chasing` function (Fig. 6) assumes that the sequence of predicates $P^{[d]}$ associated with the condition C satisfies \mathcal{M}_{dd} and \mathcal{M}_{iv} .

A process p_i executing `Decision_Chasing` first initializes local variables (line 1): current_i represents its current view, v the vertex it is about to visit (initially root), pred.d.val_i the previous value of the parameter d , and prev.nb.of.bot_i the number of occurrences of the \perp value in its previous view.

```

Function Decision_Chasing( $J_i$ ):
(1)  $current_i \leftarrow J_i; v \leftarrow root; prev.d.val_i \leftarrow +\infty; prev.nb.of.bot_i \leftarrow (f + 1);$ 
(2) while  $v$  is not a leaf do
(3)    $(current_i, direction_i) \leftarrow v.Classifier_Improver(current_i);$ 
(4)   let  $d = \max(0, \alpha(current_i) \times (\#_{\perp}(current_i) + n - L(v) - f));$ 
(5)   if  $(prev.d.val_i \neq d) \vee (prev.nb.of.bot_i \neq \#_{\perp}(current_i))$ 
(6)     then if  $P^{[d]}(current_i)$  then return  $(S(current_i))$  endif;
(7)      $prev.d.val_i \leftarrow d; prev.nb.of.bot_i \leftarrow \#_{\perp}(current_i)$ 
(8)   endif;
(9)   if  $rich_i = yes$  then  $v \leftarrow v.right$  else  $v \leftarrow v.left$  endif
(10) endwhile;
(11) if  $P^{[0]}(current_i)$  then return  $(S(current_i))$  else return  $(\top)$  endif

```

Fig. 6. The Decision_Chasing Function

Then p_i enters the tree traversal (lines 2-10). It first visits the vertex v (line 3), computes the new values of $\alpha(current_i)$ and d from its context (current values of $current_i$ and v). If something has changed (line 5) with respect to the previous predicate evaluation (either the value of d or $\#_{\perp}(current_i)$), then p_i tests $P^{[d]}(current_i)$, and if true, stops the tree traversal and returns the value $S(current_i)$ (line 6). Otherwise (line 7 and line 9), p_i updates the relevant local loop variables, and progresses to the next level of the tree in the direction indicated by the previous call to `Classifier_Improver`.

Finally, if a process has not stopped during the tree traversal (it has reached a leaf), we are in the case where there is no possibility to decide early. So, p_i checks $P^{[0]}(current_i)$ to see if it can decide. If it cannot, it sets w_i to \top .

Example. Let us consider the following example to illustrate the early decision possibility offered by `Decision_Chasing`. Let $\mathcal{V} = \{a, b, \dots\}$, $n = 6$, $f = 1$. The condition considered is $C1_f$ (see Sect. 4.3). As we will see in Sect. 7.4, this condition provides $\alpha(J) = 2, \forall J \in \mathcal{V}_f^n$.

Let us consider the input vector $I = [a, a, a, a, b, b]$. As $\#_a(I) - \#_b(I) > f$, it follows that $I \in C1_f^{[0]}$. Moreover, it is easy to see that $I \notin C1_f^{[d]}$ for $d > 0$.

Let $current_i = [a, a, a, a, \perp, b]$ be the view of p_i after its first call to `Classifier_Improver` at line 3. As $v = root$ and $L(root) = n - f$, we get $d = 2$ at line 4. Hence p_i evaluates $P^{[d]}([a, a, a, a, \perp, b]) = (\#_a(current_i) - \#_b(current_i) > f + d - \#_{\perp}(current_i))$ which is true. Consequently, p_i stops its tree traversal and exits with the value a (line 6). The fact that $P^{[2]}([a, a, a, a, \perp, b])$ is true means that there is some $I' \in C1_f^{[2]}$ such that $J \in \mathcal{I}_f$. As an example, $[a, a, a, a, a, b]$ is such an I' .

Theorem 9 *The Consensus protocol that uses Decision_Chasing (Sect. 7.1) satisfies the Validity, Agreement and Termination properties described in Sect. 2.2. Moreover, the number of shared memory accesses of its synchronization part is bounded above by $O(n \log_2(f + 1))$ for each process.*

The correctness proof appears in Appendix A.3.

7.4 From a condition to an α function

This section defines an α function that allows predicates P associated with a condition obtained from a weight function (Sect. 4.3) to satisfy \mathcal{M}_{iv} , i.e., $\forall d \geq 0, \forall J \in \mathcal{V}_f^n, \forall J' \geq J : (P^{[d]}(J) \wedge d' \leq d - \alpha(J) \text{dist}(J, J')) \Rightarrow P^{[d']}(J')$.

Theorem 10 *Let C be a condition derived from a weight function w , with the following acceptability parameters (defined in Theorem 6):*

$$\begin{aligned}
 P^{[d]}(J) &\equiv \exists a \in J, \forall b \neq a \in J, \\
 &(\#_a(J) + \#_{\perp}(J))w(a) - \#_b(J)w(b) \\
 &> (f + d) \max(w(a), w(b)); \\
 S(J) &= a \text{ s.t. } \#_a(J)w(a) = \max_{b \in J} \{\#_b(J)w(b)\}.
 \end{aligned}$$

Let $J \in \mathcal{V}_f^n$, $a = S(J)$, and M be the maximal weight associated with a value of \mathcal{V} . Let α be a function from \mathcal{V}_f^n to \mathbb{R}^+ defined as follows:

$$\bullet \alpha(J) = \begin{cases} \max_{b \in J, b \neq a} \left(\frac{w(a) + w(b)}{\max(w(a), w(b))} \right), \\ \quad \text{if } \exists x \in J : w(x) = M \\ (n - f), & \text{otherwise.} \end{cases}$$

The sequence of predicates $P^{[d]}$ satisfies the monotonicity property \mathcal{M}_{iv} with this function α .

The proof is given in Appendix A.4. The following corollary follows directly from the previous theorem and the weight-based definition of $C1_f^{[d]}$ and $C2_f^{[d]}$. As we can see, each condition provides an $\alpha(J)$ that is always ≥ 1 .

Corollary 3 *Table 1 defines α functions associated with the conditions $C1_f^{[d]}$ and $C2_f^{[d]}$ defined in Sect. 4 (M denotes the maximal weight of the elements of \mathcal{V}).*

8 Conclusion

A condition-based approach to solve the consensus problem has been introduced in [22]. It consists of identifying sets of input vectors that allow us to design a consensus protocol despite up to f process crashes. This paper has investigated the condition-based approach from a complexity perspective.

Table 1. Values of $\alpha(J)$ for $C1$ and $C2$

	$\exists x \in J : w(x) = M$	$\forall x \in J : w(x) < M$
$C1_f^{[d]}$	$\alpha(J) = 2$	Cannot occur
$C2_f^{[d]}$	$\alpha(J) = 1 + 1/n$	$\alpha(J) = (n - f)$

It has brought to the fore the fact that all conditions can be classified in a strict hierarchy of classes,

$$\mathcal{C}_f^{[\min(f, n-f)]} \subset \dots \subset \mathcal{C}_f^{[d]} \subset \dots \subset \mathcal{C}_f^{[1]} \subset \mathcal{C}_f^{[0]},$$

each class being defined by the pair (f, d) , $0 \leq d \leq f$. The class $\mathcal{C}_f^{[0]}$ contains every condition for which condition-based consensus protocols can be designed when up to f processes can crash. The degree d of a class denotes the maximum disagreement beyond f that is allowed by the conditions of this class. Actually, $(f + d)$ is the maximum distance tolerated between two processes, and the value $(f - d)$ is a measure of the difficulty of the class $\mathcal{C}_f^{[d]}$, i.e., the synchronization cost depends on $(f - d)$ ⁶. Various properties of the hierarchy have been derived in the paper. It has been shown that a class can be characterized in two equivalent but complementary ways: one is operational and helps designing protocols (acceptability), while the other is descriptive and allows analyzing the properties of a class (legality).

The paper has also introduced a generic condition-based consensus protocol that can be instantiated with any condition of any class $\mathcal{C}_f^{[d]}$. An important feature of this protocol is that the cost (measured in the number of shared memory read/write operations) of its synchronization part depends on $(f - d)$: its step complexity is bounded above by $(2n + 1) \lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$ per process. Hence, when d approaches f , the protocol becomes particularly efficient. An adaptive protocol for conditions $C \in \mathcal{C}_f$ has also been presented. Its interest resides in the fact that it never incurs additional communication cost, while reducing synchronization cost in many cases: it uses the hierarchy of classes in order to try to decide early.

Acknowledgements. We thank Nancy Lynch for discussions that led to questions considered in this paper. We thank the referees for their very constructive remarks and useful comments that helped us greatly improve the paper. We also thank Krishnamurthy Vidyasankar for his careful reading of the paper.

⁶ More precisely, the `strong_collect` synchronization primitive is close to a `snapshot` when $d = 0$, while it reduces to a simple `collect` when $d = f$.

References

1. Afek Y., Attiya H., Dolev D., Gafni E., Merritt M., Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM* **40**(4), 873–890 (1993)
2. Aguilera M.K., Toueg S., Failure Detection and Randomization: a Hybrid Approach to Solve Consensus. *SIAM Journal of Computing* **28**(3), 890–903 (1998)
3. Attiya H., Herlihy M.P., Rachman O., Atomic Snapshots Using Lattice Agreement. *Distributed Computing* **8**(3), 121–132 (1995)
4. Attiya H., Rachman O., Atomic Snapshots in $O(n \log n)$ Operations. *SIAM Journal of Computing* **27**(2), 319–340 (1998)
5. Attiya H., Welch J., *Distributed Computing: Fundamentals, Simulations and Advanced Topics*, McGraw–Hill, 451 pages (1998)
6. Ben-Or M., Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC’83)*, ACM Press, pp. 27–30, Montréal, Canada (1983)
7. Berman P., Garay J., Adaptability and the Usefulness of Hints. *6th European Symposium on Algorithms (ESA’98)*, Springer, LNCS #1461, pp. 271–282, Venice, Italy (1998)
8. Borowsky E., Gafni E., Lynch N.A., Rajsbaum S., The BG Distributed Simulation Algorithm. *Distributed Computing* **14**(3), 127–146 (2001)
9. Biran O., Moran S., Zaks S., A Combinatorial Characterization of the Distributed 1-Solvable Tasks. *Journal of Algorithms* **11**, 420–440 (1990)
10. Biran O., Moran S., Zaks S., Deciding 1-Solvability of Distributed Tasks is NP-Hard. *Proc. 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG’90)*, Springer, LNCS #484, pp. 206–220 (1990)
11. Chandra T., Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *Journal of ACM* **43**(2), 225–267 (1996)
12. Chaudhuri S., More Choices Allow More Faults: Set consensus Problems in Totally Asynchronous Systems. *Information and Computation* **105**, 132–158 (1993)
13. Dwork C., Lynch N.A., Stockmeyer L., Consensus in the Presence of Partial Synchrony. *Journal of ACM* **35**(2), 288–323 (1988)
14. Dolev D., Lynch N.A., Pinter S., Stark E.W., Weihl W.E., Reaching Approximate Agreement in the Presence of Faults. *Journal of ACM* **33**(3), 499–516 (1986)
15. Fischer M.J., Lynch N.A., Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM* **32**(2), 374–382 (1985)
16. Gafni E., Koutsoupias E., Three-Processor Tasks Are Undecidable. *SIAM Journal of Computing* **28**(3), 970–983 (1999)
17. Herlihy M.P., Rajsbaum S., On the Decidability of Distributed Decision Tasks. *Proc. 29th ACM Symposium on the Theory of Computing (STOC’97)*, ACM Press, pp. 589–598 (1997)
18. Herlihy M.P., Rajsbaum S., New Perspectives in Distributed Computing. *Invited Talk, Proc. 24th Int. Symposium on Mathematical Foundations of Computer Science (MFCS’99)*, Springer, LNCS #1672, pp. 170–186 (1999)
19. Inoue M., Chen W., Masuzawa T., Tokura N., Linear-Time Snapshot Using Multi-Writer Multi-Reader Registers. *Proc. 8th Int. Workshop on Distributed Algorithms (WDAG’94)*, Springer, LNCS #857, pp. 130–140 (1994)
20. Israeli A., Shaham A., Shirazi A., Linear-Time Snapshot Protocols for Unbalanced Systems. *Proc. 7th Int. Workshop on Distributed Algorithms*, Springer, LNCS #725, pp. 26–38 (1993)
21. Lynch N.A., *Distributed Algorithms*. Morgan Kaufmann Pub., San Francisco, CA, 872 pages (1996)

22. Mostéfaoui A., Rajsbaum S., Raynal M., Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems. *Proc. 33rd ACM Symposium on Theory of Computing (STOC'01)*, ACM Press, pp. 153–162, Hersonissos, Crete, Greece (2001)
23. Mostéfaoui A., Rajsbaum S., Raynal M., Roy M., A Hierarchy of Conditions for Consensus Solvability. *Proc. 20th ACM Symposium on Principles of Distributed Computing (PODC'01)*, ACM Press, pp. 151–160, Newport, RI (2001)
24. Mostéfaoui A., Rajsbaum S., Raynal M., Roy M., Condition-Based Protocols for Set Agreement Problems, *Proc. 16th Int. Symp. on Distributed Computing (DISC'02)*, Springer, LNCS #2508, pp. 48–62 (2002)
25. Mostéfaoui A., Rajsbaum S., Raynal M., Roy M., Efficient Condition-Based Consensus. *Proc. 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO'01)*, Carleton Scientific Press, pp. 275–293, Vall de Nùria, Catalonia, Spain (2001)
26. Mostéfaoui A., Raynal M., Tronel F., The Best of Both Worlds: a Hybrid Approach to Solve Consensus. *Proc. IEEE Int. Conf. on Dependable Systems and Networks (DSN'00, previously FTCS)*, pp. 513–522 (2000)
27. Taubenfeld G., Katz S., Moran S., Impossibility Results in the Presence of Multiple Faulty Processes. *Information and Computation* **113**(2), 173–198 (1994)
28. Taubenfeld G., Moran S., Possibility and Impossibility Results in a Shared Memory Environment. *Acta Informatica* **35**, 1–20 (1996)

A Proofs

A.1 Weight-based definition of conditions

Theorem 6 $\forall d \geq 0$, any function w from \mathcal{V} to \mathbb{R}^+ defines a condition $C \in \mathcal{C}_f^{[d]}$, with the following parameters:

$$P(J) \equiv \exists a \in J : \forall b \in J, b \neq a, \\ (\#_a(J) + \#_{\perp}(J))w(a) - \#_b(J)w(b) \\ > (f + d) \max(w(a), w(b));$$

$$S(J) = a \text{ s.t. } \#_a(J)w(a) = \max\{\#_b(J)w(b) : b \in J\} \\ (\text{the value distinguished by } P(J)).$$

Proof. Let w be a weight function, (f, d) a pair of integers, and C the corresponding condition derived from the formula. We show that C is (f, d) -acceptable with parameters (P, S) as defined in the theorem.

- Property $T_{C \rightarrow P}$: $I \in C \Rightarrow \forall J \in \mathcal{I}_f : P(J)$.
Let $I \in C$. Let a be such that $\forall b \in I, b \neq a$, we have (1) $w(a)\#_a(I) - w(b)\#_b(I) > (f + d) \max(w(a), w(b))$. Let J be a view of \mathcal{I}_f . Then, (2) $\#_a(J) \geq \#_a(I) - \#_{\perp}(J)$ and (3) $\#_b(J) \leq \#_b(I), \forall b$. Combining (1), (2) and (3) gives: $w(a)(\#_a(J) + \#_{\perp}(J)) - w(b)\#_b(J) > (f + d) \max(w(a), w(b))$, i.e., $P(J)$ holds.
- Property $V_{P \rightarrow S}$: $\forall I \in \mathcal{V}^n : \forall J \in \mathcal{I}_f : P(J) \Rightarrow (S(J) = \text{a non-}\perp \text{ value of } J)$. Follows directly from the definition of S , as J contains at least $(n - f)$ non- \perp values.
- Property $A_{P \rightarrow S}^{[d]}$:

$$\forall I \in \mathcal{V}^n : \forall J1, J2 \in \mathcal{I}_f \text{ s.t. } P(J1) \wedge P(J2) : \\ ((J1 \leq J2) \vee (\#_{\perp}(J1) + \#_{\perp}(J2) \leq f + d)) \\ \Rightarrow S(J1) = S(J2).$$

Let $J1$ and $J2$ be two views of some \mathcal{I}_f such that both satisfy P . We consider two cases:

1. $J1 \leq J2$.

Let $a = S(J2)$. Since $J1 \leq J2$, we have: $\#_a(J1) \geq \#_a(J2) - (\#_{\perp}(J1) - \#_{\perp}(J2))$ and $\forall b : \#_b(J1) \leq \#_b(J2)$. Then, for any b in $J1$:

$$(\#_a(J1) + \#_{\perp}(J1))w(a) - \#_b(J1)w(b) \\ \geq (\#_a(J2) - (\#_{\perp}(J1) - \#_{\perp}(J2))) \\ + \#_{\perp}(J1)w(a) - \#_b(J2)w(b) \\ = (\#_a(J2) + \#_{\perp}(J2))w(a) - \#_b(J2)w(b) \\ > (f + d) \max(w(a), w(b)).$$

It follows that a is the value of $J1$ that is distinguished by $P(J1)$. Hence, $S(J1) = a$.

2. $\#_{\perp}(J1) + \#_{\perp}(J2) \leq f + d$.

Suppose that $a = S(J1)$ and $b = S(J2)$ are different. First, let us notice the following inequality: $\#_a(J1) + \#_{\perp}(J1) > f + d$, since $P(J1)$ holds and w is a positive function. Similarly, we have for b : $\#_b(J2) + \#_{\perp}(J2) > f + d$. Now we can check that $a \in J2$: $\#_a(J2) \geq \#_a(J1) - \#_{\perp}(J2)$, and, using the above inequality, $\#_a(J2) > (f + d - \#_{\perp}(J1)) - \#_{\perp}(J2)$. Then, as $\#_{\perp}(J1) + \#_{\perp}(J2) \leq f + d$, we get that $\#_a(J2) > 0$ (the same reasoning on b and $J1$ gives $\#_b(J1) > 0$).

Let us now apply the definition of P to $J1$ (resp. $J2$) with the pair (a, b) (resp. (b, a)). We get:

$$(\#_a(J1) + \#_{\perp}(J1))w(a) - \#_b(J1)w(b) > \\ (f + d) \max(w(a), w(b)) \quad (1)$$

$$(\#_b(J2) + \#_{\perp}(J2))w(b) - \#_a(J2)w(a) > \\ (f + d) \max(w(a), w(b)) \quad (2)$$

Summing up these two inequalities we get:

$$w(a)(\#_a(J1) - \#_a(J2) + \#_{\perp}(J1)) \\ + w(b)(\#_b(J2) - \#_b(J1) + \#_{\perp}(J2)) \\ > 2(f + d) \max(w(a), w(b)).$$

Let us notice that both quantities $(\#_a(J1) - \#_a(J2) + \#_{\perp}(J1))$ and $(\#_b(J2) - \#_b(J1) + \#_{\perp}(J2))$ are non negative. This comes from the fact that $J1$ and $J2$ are views of the same input vector I , and $\forall J, J' \in \mathcal{I}_f, \forall x \in I : \#_x(J') \leq \#_x(I) \leq \#_x(J) + \#_{\perp}(J)$. It follows that the left part of the previous inequality is bounded above by: $\max(w(a), w(b)) (\#_a(J1) - \#_a(J2) + \#_{\perp}(J1) + \#_b(J2) - \#_b(J1) + \#_{\perp}(J2))$. Since $\#_a(J2) \geq \#_a(J1) - \#_{\perp}(J2)$ and $\#_b(J1) \geq \#_b(J2) - \#_{\perp}(J1)$, this quantity is $\leq 2 \max(w(a), w(b)) (\#_{\perp}(J1) + \#_{\perp}(J2))$, which in turn is $\leq 2(f + d) \max(w(a), w(b))$. It follows that $2(f + d) \max(w(a), w(b)) > 2(f + d) \max(w(a), w(b))$: a contradiction. Hence, our initial assumption $a \neq b$ is false.

□*Theorem 6*

A.2 Properties of the tree traversal

Lemma 4 Assume there is an input vector I such that processes p_i invoke $\text{strong_collect}_f^{[d]}$ with views $J_i \in \mathcal{I}_f$. For every vertex v and process p_i :

1. $L(v) \leq |J_{i,v}| \leq H(v)$ when $p_i \in \text{visited}(v)$,
2. $|\text{lub}(\{J_{i,v} : p_i \in \text{visited}(v)\})| \leq H(v)$,
3. $J_{i,v} < J_{j,u}$ whenever $H(v) < L(u) \wedge p_i \in \text{visited}(v) \wedge p_j \in \text{visited}(u)$,
4. $J_{i,v} = J_{j,v}$ whenever $L(v) = H(v) \wedge p_i, p_j \in \text{visited}(v)$.

Proof. Let us first observe that, as strong_collect uses $\text{classifier_improver}$, we have to show that the input assertion associated with this abstraction (namely, $|\text{lub}(\{J_i\})| \leq H(v)$) is satisfied when $v.\text{classifier_improver}$ is invoked. But this input assertion is exactly item 2 of the lemma. Hence, it is part of the proof of the lemma. The structure of the rest of the proof is the following. (i) We first prove simultaneously items 1 and 2 by induction on the tree. (ii) Then, we prove items 3 and 4

(i) Proof of items 1 and 2.

The proof is by induction. Let us first consider the root. Since a process p_i provides strong_collect with an initial view J_i with at least $(n - f)$ non- \perp entries, item 1 is trivially satisfied for the root. Due to $H(\text{root}) = n$, item 2 is also trivially satisfied.

Considering a non-leaf vertex v , let us now assume that item 1 (namely, $L(v) \leq |J_{i,v}| \leq H(v)$ when $p_i \in \text{visited}(v)$) and item 2 (namely, $|\text{lub}(\{J_{i,v} : p_i \in \text{visited}(v)\})| \leq H(v)$) of the lemma are satisfied. We prove that they are satisfied by both of the children of v .

- Case $v.\text{left}$.

- Due to Specification 6.2(5) applied to v , we have $|\text{lub}(\{J_{i,v.\text{left}} : \text{rich}_{i,v} = \text{no}\})| \leq H(v.\text{left})$. By definition,

$$\text{visited}(v.\text{left}) = \{p_i \in \text{visited}(v) \text{ s. t. } \text{rich}_{i,v} = \text{no}\},$$

we conclude that item 2 is satisfied for $v.\text{left}$.

- Let $p_i \in \text{visited}(v.\text{left})$. From the previous conclusion, we also have $|J_{i,v.\text{left}}| \leq H(v.\text{left})$. Moreover, due to Specification 6.2(1) applied to v , we have $J_{i,v} \leq J_{i,v.\text{left}}$ ($J_{i,v.\text{left}}$ is the result of $v.\text{classifier_improver}$). We also have (due to the induction hypothesis) $|J_{i,v}| \geq L(v)$, and (due to the labeling definition) $L(v) = L(v.\text{left})$. Combining these inequalities, we get $L(v.\text{left}) \leq |J_{i,v.\text{left}}| \leq H(v.\text{left})$ which proves item 1 for $v.\text{left}$.

- Case $v.\text{right}$.

- From item 2 of the lemma applied to v (induction assumption), we get $|\text{lub}(\{J_{i,v} : p_i \in \text{visited}(v)\})| \leq H(v)$. Moreover, due to Specification 6.2(4) applied to vertex v , we get

$$\begin{aligned} \text{lub}(\{J_{i,v.\text{right}} : p_i \in \text{visited}(v.\text{right})\}) &\leq \\ &\text{lub}(\{J_{i,v} : p_i \in \text{visited}(v)\}). \end{aligned}$$

Going from sets to their cardinal numbers, we get:

$$|\text{lub}(\{J_{i,v.\text{right}} : p_i \in \text{visited}(v.\text{right})\})| \leq$$

$$|\text{lub}(\{J_{i,v} : p_i \in \text{visited}(v)\})|.$$

Finally, as $H(v) = H(v.\text{right})$ (from the labeling definition), item 2 follows for $v.\text{right}$.

- Due to Specification 6.2(2) applied to v , and the fact that $\text{rich}_{i,v} = \text{yes}$, we have $|J_{i,v.\text{right}}| > H(v.\text{left})$. Moreover, $H(v.\text{left}) = L(v.\text{right}) - 1$ due to the labeling definition. It follows that $|J_{i,v.\text{right}}| \geq L(v.\text{right})$. On the other side, $|J_{i,v.\text{right}}| \leq H(v.\text{right})$ follows from the previous proof of item 2. Hence, item 1 is satisfied for $v.\text{right}$.

(ii) Proof of items 3 and 4.

For item 3, let us consider the vertex w that is the common ancestor of u and v , such that $\text{rich}_{i,w} = \text{no}$ and $\text{rich}_{j,w} = \text{yes}$ (i.e., w is the vertex where the traversals of p_i and p_j become different). As $H(v) < L(u)$, the vertex w exists and is distinct from u and v , and there are paths from $w.\text{left}$ to v and from $w.\text{right}$ to u . Let us consider the left side of w . Let us consider a path in the tree from an ancestor x of v to v . From Specification 6.2(4) and the fact that a process progresses downwards in the tree, we have $J_{i,v} \leq \text{lub}(\{J_{k,x} : p_k \in \text{visited}(x)\})$. Taking $x = w.\text{left}$, we have from the previous claim:

$$\begin{aligned} J_{i,v} &\leq \text{lub}(\{J_{k,w.\text{left}} : p_k \in \text{visited}(w.\text{left})\}) \\ &= \text{lub}(\{J_{k,w.\text{left}} : p_k \in \text{visited}(w.\text{left}) \\ &\quad \wedge \text{rich}_{k,w} = \text{no}\}). \end{aligned}$$

Let us now consider the right side of w . Applying iteratively Specifications 6.2(3) and 6.2(1) to the vertices of the path from $w.\text{right}$ to u , we get $J_{j,w.\text{right}} \leq J_{j,u}$. Finally, from Specification 6.2(3), we have $\forall k : \text{rich}_{k,w} = \text{no} : J_{k,w.\text{left}} < J_{j,w.\text{right}}$. Combining the previous inequalities we get

$$\begin{aligned} J_{i,v} &\leq \text{lub}(\{J_{k,w.\text{left}} : p_k \in \text{visited}(w.\text{left}) \\ &\quad \wedge \text{rich}_{k,w} = \text{no}\}) \\ &< J_{j,w.\text{right}} \leq J_{j,u}, \end{aligned}$$

which proves item 3.

The proof of 4 follows from item 1 of the lemma (which implies $|J_{i,v}| = |J_{j,v}| = L(v) = H(v)$), and from item 2 (which implies $J_{i,v} = J_{j,v}$).

□_{Lemma 4}

A.3 Correctness proof of the early stopping protocol

The core of the proof is the following lemma that shows that the Decision_Chasing function cannot violate the agreement property. The proof assumes $\alpha(J) \geq 1, \forall J \in \mathcal{V}_f^n$.

Lemma 7 Let p_i and p_j be two processes that exit Decision_Chasing with w_i and w_j , respectively. Then, $(w_i \neq \top \wedge w_j \neq \top) \Rightarrow (w_i = w_j)$.

Proof. Let us consider two processes p_i and p_j such that $w_i \neq \top$ and $w_j \neq \top$. Hence, p_i (resp. p_j) exited Decision_Chasing at line 6 or 11 because $P^{[d1]}(J1)$ (respectively $P^{[d2]}(J2)$) was true with $w_i = S(J1)$ (resp. $w_j = S(J2)$). Let $v1$ (resp. $v2$) be the corresponding vertex of the tree (an

interior vertex if the exit is at line 6 or a leaf if the exit is at line 11). The proof proceeds by case analysis. We consider three cases.

Case 1: The vertices $v1$ and $v2$ belong to two distinct paths from the root to leaves. When considering $v1$ and $v2$, without loss of generality, let $v1$ be the leftmost vertex. Moreover, let $d = \min(d1, d2)$.

The monotonicity property \mathcal{M}_{dd} ensures $P^{[d2]}(J2) \Rightarrow P^{[d]}(J2)$ and $P^{[d1]}(J1) \Rightarrow P^{[d]}(J1)$. Moreover, we have $H(v1) < L(v2)$ (labeling of the tree), $p_i \in \text{visited}(v1)$ and $p_j \in \text{visited}(v2)$ (as each process visited the corresponding vertex). From this, we can conclude (due to Item 3 of Lemma 4) that $J1 \leq J2$. Finally, combining $J1 \leq J2 \wedge P^{[d]}(J2) \wedge P^{[d]}(J1)$ and Property $A_{P \rightarrow S}^{[d]}$, we get $S(J1) = S(J2)$, i.e., $w_i = w_j$.

Case 2: The vertices $v1$ and $v2$ are the same vertex v . We consider two subcases.

- (i) v is not a leaf. Without loss of generality let us assume $d1 = \min(d1, d2)$. Due to the classifier/improver tree (Item 1 of Lemma 4) we have $\#_{\perp}(J2) \leq (n - L(v))$, from which we get: $\#_{\perp}(J1) + \#_{\perp}(J2) \leq \#_{\perp}(J1) + (n - L(v))$. Moreover (definition of $d1$ at line 4), we have $d1 \geq \alpha(J1) \times (\#_{\perp}(J1) + n - L(v) - f)$, from which we get $((d1/\alpha(J1)) + f) \geq (\#_{\perp}(J1) + n - L(v))$. Hence, we obtain $\#_{\perp}(J1) + \#_{\perp}(J2) \leq (d1/\alpha(J1)) + f$. As $\alpha(J1) \geq 1$ (by assumption), we can conclude $\#_{\perp}(J1) + \#_{\perp}(J2) \leq (d1 + f)$. Finally, combining this inequality with $P^{[d1]}(J2) \wedge P^{[d1]}(J1)$ and Property $A_{P \rightarrow S}^{[d1]}$, we get $S(J1) = S(J2)$, i.e., $w_i = w_j$.
- (ii) v is a leaf. In that case, p_i and p_j exited at line 11.
 - If v is not the rightmost leaf, then $J1 = J2$ (Item 4 of Lemma 4), and $S(J1) = S(J2)$.
 - If v is the rightmost leaf, then each of $J1$ and $J2$ has at most $\lfloor f/2 \rfloor$ entries equal to \perp . It follows that we then have:

$$P^{[0]}(J1) \wedge P^{[0]}(J2) \wedge \#_{\perp}(J1) + \#_{\perp}(J2) \leq f,$$

which, combined with $A_{P \rightarrow S}^{[0]}$, provides $S(J1) = S(J2)$.

Case 3: The vertices $v1$ and $v2$ belong to the same path from the root to a leaf. Without loss of generality, let $v1$ be the vertex closest to the root. Let $J' = \text{lub}(\{J1, J2\})$. We consider three cases.

- $d1 \leq d2$.
In that case we have $P^{[d1]}(J1)$, $P^{[d1]}(J2)$ and $\#_{\perp}(J1) + \#_{\perp}(J2) \leq \#_{\perp}(J1) + (n - L(v2))$. As $L(v1) \leq L(v2)$ ($v1$ is closer to the root than $v2$), $\#_{\perp}(J1) + \#_{\perp}(J2) \leq \#_{\perp}(J1) + n - L(v1)$. From then on, the proof is the same as the end of the proof of Case 2 (i).
- $d1 > d2$ and $\#_{\perp}(J1) - \#_{\perp}(J') < (d1 - d2)/\alpha(J1)$.
We have $\text{dist}(J1, J') < (d1 - d2)/\alpha(J1)$. Due to the monotonicity property \mathcal{M}_{iv} we have $P^{[d2]}(J')$. Using that $P^{[d2]}(J') \wedge P^{[d2]}(J2) \wedge J2 \leq J'$, we get $S(J2) = S(J')$ from $A_{P \rightarrow S}^{[d2]}$.
As $d1 > d2$, Due to the monotonicity property \mathcal{M}_{dd} , we also have $P^{[d2]}(J1)$. As in the previous situation, we get

$P^{[d2]}(J') \wedge P^{[d2]}(J1) \wedge J1 \leq J'$, from which we conclude $S(J1) = S(J')$ (due to $A_{P \rightarrow S}^{[d2]}$). It follows that $S(J1) = S(J2)$.

- $d1 > d2$ and $\#_{\perp}(J1) - \#_{\perp}(J') \geq (d1 - d2)/\alpha(J1)$.
Let $J1'$ be such that (1) $\#_{\perp}(J1) = \#_{\perp}(J1') + (d1 - d2)/\alpha(J1)$, and (2) each entry which is equal to \perp in $J1$ and different from \perp in $J1'$ is equal to the corresponding entry of $J2$. Due to $\#_{\perp}(J1) - \#_{\perp}(J') \geq (d1 - d2)/\alpha(J1)$, such a view does exist.
 - Let us first consider $J1$ and $J1'$. As $P^{[d2]}(J1)$ holds (monotonicity \mathcal{M}_{dd}), $P^{[d2]}(J1')$ holds (monotonicity \mathcal{M}_{iv}) and $J1 < J1'$ (construction of $J1'$), we get that $S(J1) = S(J1')$ from $A_{P \rightarrow S}^{[d2]}$.
 - Let us now consider $J1'$ and $J2$. From (1) $\#_{\perp}(J2) \leq f$, (2) $\#_{\perp}(J2) \leq (n - L(v2)) \leq (n - L(v1))$ ($J2$ has been obtained at vertex $v2$ and $v1$ is an ancestor of $v2$), and (3) the fact that $J1$ and $J1'$ differ in exactly $(d1 - d2)/\alpha(J1)$ entries (those that are equal to \perp in $J1$ and different in $J1'$), we get

$$\begin{aligned} \#_{\perp}(J1') + \#_{\perp}(J2) &\leq \\ \#_{\perp}(J1) - ((d1 - d2)/\alpha(J1)) + (n - L(v1)). \end{aligned}$$

As $d1 \geq \alpha(J1) \times (\#_{\perp}(J1) + n - L(v1) - f)$ (definition of $d1$, line 4), we have

$$\begin{aligned} \#_{\perp}(J1') + \#_{\perp}(J2) &\leq \\ (d1/\alpha(J1)) + f - ((d1 - d2)/\alpha(J1)). \end{aligned}$$

As $\alpha(J1) \geq 1$ (assumption), it follows that $\#_{\perp}(J1') + \#_{\perp}(J2) \leq (d2 + f)$. From this inequality, $P^{[d2]}(J1')$, $P^{[d2]}(J2)$ and $A_{P \rightarrow S}^{[d2]}$, we get $S(J1') = S(J2)$. Finally, combining $S(J1) = S(J1')$ and $S(J1') = S(J2)$, we get $S(J1) = S(J2)$.

□_{Lemma 7}

Theorem 9

The *Consensus* protocol that uses *Decision.Chasing* satisfies the *Validity*, *Agreement* and *Termination* properties described in Sect. 2.2. Moreover, the number of shared memory accesses of its synchronization part is bounded above by $n \log_2(f + 1)$ for each process.

Proof.

Validity: A decided value is a proposed value. This property follows from the code (the only values manipulated by the protocol are proposed values plus \perp) and Property $V_{P \rightarrow S}$.

Agreement: No two processes decide different values. We proceed by case analysis.

Let us first consider two processes p_i and p_j that decide in line 6. They get $w_i \neq \top$ and $w_j \neq \top$ from the *Decision.Chasing* function. From Lemma 7, we have $w_i = w_j$.

Let us now consider the case of a process p_i that decides at line 6 while another process p_j decides at line 8. Process p_j actually decides the value w_k deposited in W by another process p_k . Hence, p_k decided $w_k \neq \top$ at line 6. It follows from the previous item that $w_i = w_k$.

Let us consider two processes p_i and p_j that decide at line 11. In that case, both processes apply the same deterministic

function to the same array $[a_1, \dots, a_n]$. Consequently, they decide the same value.

Finally, let us show that it is not possible for a process p_i to decide at line 6 or 8, while another process p_j decides at line 11. If p_i decides at line 6, we have $W[i] \neq \top$. Similarly, if p_i decides at line 8, there is some p_k such that $W[k] \neq \top$. As the array W is initialized to $[\perp, \dots, \perp]$, it follows that before p_j exits the **repeat** loop (lines 7-9), there exists a $W[k] \neq \perp$ or \top and consequently p_j exits at line 8.

Termination: If (1) $I \in \mathcal{C}_f$ and no more than f processes crash, or (2) all processes are correct, or (3) a process decides, then each correct process decides. The proof is the same as in Theorem 8, using the fact that **Decision_Chasing** is synchronization, and Property $T_{C \rightarrow P}$.

Step Complexity. Follows directly from the fact that the depth of the tree is $\lceil \log_2(\lceil f/2 \rceil + 1) \rceil$, and the fact that a call to **Classifier_Improver** costs at most $(2n + 1)$ shared memory read/write operations. $\square_{\text{Theorem 9}}$

A.4 From a condition to an α function

Theorem 10 Let C be a condition derived from a weight function w , with the following acceptability parameters (defined in Theorem 6):

$$\begin{aligned} P^{[d]}(J) &\equiv \exists a \in J, \forall b \neq a \in J, \\ &(\#_a(J) + \#_{\perp}(J))w(a) - \#_b(J)w(b) \\ &> (f + d) \max(w(a), w(b)); \\ S(J) &= a \text{ s.t. } \#_a(J)w(a) = \max_{b \in J} \{\#_b(J)w(b)\}. \end{aligned}$$

Let $J \in \mathcal{V}_f^n$, $a = S(J)$, and M be the maximal weight associated with a value of \mathcal{V} . Let α be a function from \mathcal{V}_f^n to \mathbb{R}^+ defined as follows:

- $\alpha(J) = \max_{b \in J, b \neq a} \left(\frac{w(a) + w(b)}{\max(w(a), w(b))} \right)$,
if $\exists x \in J : w(x) = M$.
- $\alpha(J) = (n - f)$, otherwise.

The sequence of predicates $P^{[d]}$ satisfies the monotonicity property \mathcal{M}_{iv} with this function α .

Proof. Let J be a view such that $P^{[d]}(J)$ holds. Our aim is to show that if (1) $J' \geq J$, and (2) $\text{dist}(J, J') \leq \frac{d-d'}{\alpha(J)}$, then $P^{[d']}(J')$ holds. Let us notice the following relations directly derived from the definitions of J and J' :

- (R1): $\text{dist}(J, J') = \#_{\perp}(J) - \#_{\perp}(J')$.
- (R2): $\forall x : \#_x(J) \leq \#_x(J')$.
- (R3): $\forall x : \#_x(J') \leq \#_x(J) + \#_{\perp}(J) - \#_{\perp}(J')$.

The proof is made up of four cases.

Case 1: $\exists x \in J : w(x) = M$ and J and J' include the same set of values of \mathcal{V} .

We show that a (the value that makes $P^{[d]}(J)$ hold) also makes $P^{[d']}(J')$ hold. For any $b \in J', b \neq a$, let us consider the following expression $(\#_a(J') + \#_{\perp}(J'))w(a) - \#_b(J')w(b)$. It is:

$$\begin{aligned} &\geq \#_a(J)w(a) + \#_{\perp}(J')w(a) - \#_b(J')w(b) \quad (\text{due to R2}) \\ &\geq \#_a(J)w(a) + (\#_{\perp}(J)w(a) - \text{dist}(J, J')w(a)) - \end{aligned}$$

$$\begin{aligned} &\#_b(J')w(b) \quad (\text{due to R1}) \\ &\geq \#_a(J)w(a) + (\#_{\perp}(J)w(a) - \text{dist}(J, J')w(a)) - \\ &\quad (\#_b(J)w(b) + \text{dist}(J, J')w(b)) \quad (\text{due to R3}) \\ &> (f + d) \max(w(a), w(b)) - \text{dist}(J, J')(w(a) + w(b)) \\ &\quad (\text{as } P^{[d]}(J) \text{ holds}) \\ &> (f + d' + \alpha(J) \text{dist}(J, J')) \max(w(a), w(b)) - \\ &\quad \text{dist}(J, J')(w(a) + w(b)) \quad (\text{replacement of } d) \\ &> (f + d') \max(w(a), w(b)) + \text{dist}(J, J')(\alpha(J) \times \\ &\quad \max(w(a), w(b)) - (w(a) + w(b))) \quad (\text{rewriting}) \\ &> (f + d') \max(w(a), w(b)) \quad (\text{due to the definition of } \alpha(J)). \end{aligned}$$

Hence, $(\#_a(J') + \#_{\perp}(J'))w(a) - \#_b(J')w(b) > (f + d') \times \max(w(a), w(b))$, i.e., $P^{[d']}(J')$ holds.

Case 2: $\exists x \in J : w(x) = M$ and J' contains values b' that are not in J , and a (the value that makes $P^{[d]}(J)$ hold) is such that $w(a) = M$. The proof of $P^{[d']}(J')$ for the values b that are in J and in J' is the same as in Case 1.

Let b' be such that $b' \in J' \wedge b' \notin J$. We want to show that $(\#_a(J') + \#_{\perp}(J'))w(a) - \#_{b'}(J')w(b') > (f + d') \max(w(a), w(b')) = (f + d') M$. Let us consider the expression $(\#_a(J') + \#_{\perp}(J'))w(a) - \#_{b'}(J')w(b')$. It is:

$$\begin{aligned} &\geq (\#_a(J) + \#_{\perp}(J) - \text{dist}(J, J'))w(a) - \#_{b'}(J')w(b') \\ &\quad (\text{due to R1 and R2}) \\ &> (f + d) M - \text{dist}(J, J')w(a) - \#_{b'}(J')w(b') \\ &\quad (\text{as } P^{[d]}(J) \Rightarrow (\#_a(J) + \#_{\perp}(J))w(a) > (f + d) M) \\ &> (f + d' + \alpha(J) \text{dist}(J, J')) M - \text{dist}(J, J')w(a) - \\ &\quad \#_{b'}(J')w(b') \quad (\text{definition of } d') \\ &> (f + d') M + \alpha(J) \text{dist}(J, J') M - \text{dist}(J, J')M - \\ &\quad \#_{b'}(J')w(b') \\ &> (f + d') M + \alpha(J) \text{dist}(J, J') M - \text{dist}(J, J')(M + \\ &\quad w(b')) \quad (\text{as } \#_{b'}(J') \leq \text{dist}(J, J')) \\ &> (f + d') M \quad (\text{due to the definition of } \alpha(J)). \end{aligned}$$

Case 3: $\exists x \in J : w(x) = M$ and J' contains values b' that are not in J , and a (the value that makes $P^{[d]}(J)$ hold) is such that $w(a) < M$.

Let us consider the worst case where $w(b') = M$. The proof of this case is the same as Case 1, by replacing b' with a value b such that $w(b) = M$.

Case 4: $\forall x \in J : w(x) < M$.

In this case $\alpha(J) = (n - f)$, which means that $\text{dist}(J, J') < 1$. In other words, the disc of views centered at J and with radius $\text{dist}(J, J')$ is reduced to J , and the property \mathcal{M}_{iv} is trivially satisfied. $\square_{\text{Theorem 10}}$

Achour Mostefaoui is currently Assistant Professor at the Computer Science Department of the University of Rennes, France. He received his Engineer Degree in Computer Science in 1990 from the University of Algiers, and a Ph.D. in Computer Science in 1994 from the University of Rennes, France. His research interests include fault-tolerance in distributed systems, group communication, consistency in DSM systems and distributed checkpointing. Achour Mostefaoui has published more than 60 scientific publications and served as a reviewer for more than 20 major journals and conferences. Moreover, Achour Mostefaoui is heading the software engineer degree of the University of Rennes.

Sergio Rajsbaum received a degree in Computer Engineering from the National Autonomous University of Mexico (UNAM) in 1985, and a PhD in the Computer Science from the Technion, Israel, in 1991. Since then he has been a member of the Institute of Mathematics at UNAM. He has been a visiting scientist at the Laboratory for Computer Science of MIT, and the Cambridge Research Laboratory of HP. He was chair of the program committee for Latin American Theoretical Informatics LATIN2002, and for ACM Principles of Distributed Computing PODC03. His research interests are in the theory of distributed computing, especially issues related to coordination, complexity and computability. He has also published in graph theory and algorithms. He runs the Distributed Computing Column of SIGACT News, the newsletter the ACM Special Interest Group on Algorithms and Computation Theory.

Michel Raynal has been a professor of computer science since 1981. At IRISA (CNRS-INRIA-University joint computing research laboratory located in Rennes), he founded a research group on Distributed Algorithms in 1983. His research interests include distributed algorithms, distributed computing systems, networks and dependability. His main interest lies in the fundamental principles that underly the design and the construction of distributed computing systems. He has been Principal Investigator of a number of research grants in these areas, and has been invited by many universities to give lectures about operating systems and distributed algorithms in Europe, south and north America, Asia and Africa. He is serving as an editor for several international journals.

Professor Michel Raynal has published more than 75 papers in journals (Acta Informatica, Distributed Computing, Comm. of the ACM, Information and Computation, Journal of Computer and System Sciences, JPDC, IEEE Transactions on Computers, IEEE Transactions on SE, IEEE Transactions on KDE, IEEE Transactions on TPDS, IEEE Computer, IEEE Software, IPL, PPL, Theoretical Computer Science, Real-Time Systems Journal, The Computer Journal, etc.); and more than 160 papers in conferences (ACM STOC, ACM PODC, ACM SPAA, IEEE ICDCS, IEEE DSN, DISC, IEEE IPDPS, Europar, FST&TCS, IEEE SRDS, etc.). He has also written seven books devoted to parallelism, distributed algorithms and systems (MIT Press and Wiley). Michel Raynal has served in program committees for more than 70 international conferences (including ACM PODC, DISC, ICDCS, DSN, SRDS, etc.) and chaired the program committee of more than 15 international conferences. He is currently serving as the chair of the steering committee leading the DISC symposium series. Michel Raynal got the IEEE ICDCS best paper Award three times in a row: 1999, 2000 and 2001.

Matthieu Roy is currently PhD student at IRISA, a CNRS-INRIA-University joint research center in Rennes, France, advised by Pr. Michel Raynal. His research interests include fault tolerant distributed computing, distributed algorithms and their applications to distributed databases and systems.