

# A Hierarchy of Conditions for Consensus Solvability

[Extended Abstract] \*

Achour Mostefaoui<sup>†</sup>  
IRISA, Campus Beaulieu  
35042 Rennes Cedex  
France

Sergio Rajsbaum<sup>‡</sup>  
Compaq CRL  
One Cambridge Center  
02142 Cambridge, MA

Michel Raynal<sup>§</sup>  
IRISA, Campus Beaulieu  
35042 Rennes Cedex  
France

Matthieu Roy<sup>¶</sup>  
IRISA, Campus Beaulieu  
35042 Rennes Cedex  
France

## ABSTRACT

In a previous paper we introduced the *condition-based* approach, consisting 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, and characterized this set of conditions,  $\mathcal{C}_f^{w,k}$ . Here we investigate  $\mathcal{C}_f^{w,k}$  from the complexity perspective, and show that this class consists of a hierarchy of classes of conditions,  $\mathcal{C}_f^{[d]}$ , where  $d$ ,  $0 \leq d \leq f$ , is the *degree* of the condition, each one strictly contained in the previous one. The value  $f - d$  represents the “*difficulty*” of the class  $\mathcal{C}_f^{[d]}$ : we present a generic condition-based protocol that can be instantiated with any  $C \in \mathcal{C}_f^{[d]}$ , and solve consensus with  $(2n + 1) \lceil \log_2(\lceil (f - d)/2 \rceil + 1) \rceil$  shared memory read/write operations per process. For each  $d$  we present two natural conditions,  $C1_f^{[d]}$  and  $C2_f^{[d]}$ , that might be useful in practice, and we use them to show that the class containments stated above are strict. Various properties of the hierarchy are also derived. Mainly, 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.

## Keywords

Asynchronous Shared Memory, Consensus, Fault-Tolerance, Snapshot, Step Complexity.

\* A full version of this paper is available in [23].

<sup>†</sup>achour@irisa.fr

<sup>‡</sup>Sergio.Rajsbaum@compaq.com. On leave from Instituto de Matemáticas, UNAM, Mexico. rajsbaum@math.unam.mx

<sup>§</sup>raynal@irisa.fr

<sup>¶</sup>mroy@irisa.fr

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

PODC'01, August 26-29, 2001, Newport, Rhode Island.

Copyright 2001 ACM ...\$5.00.

## 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 (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 for probabilistic solutions (e.g., [6]), or for approximate solutions ( $\epsilon$ -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, 25], 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 introduced a new *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 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. Safe means that the protocol guarantees agreement and validity, whether the proposed input vector is allowed by the condition or not. In addition, we would like the protocols to make the “best effort” to terminate (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 that in some real system 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).

In [22], we characterized the conditions that admit a consensus protocol with the above properties. That is, we described a set of conditions, denoted here  $\mathcal{C}_f^{wk}$ , and proved that there is a consensus protocol for a condition  $C$  if and only if  $C \in \mathcal{C}_f^{wk}$ . We presented two equivalent combinatorial descriptions of the class  $\mathcal{C}_f^{wk}$ , and described two natural conditions  $C1$  and  $C2$  in  $\mathcal{C}_f^{wk}$  that might be useful in practice, and proved them to be maximal (they cannot be extended). The class  $\mathcal{C}_f^{wk}$  is quite rich, since it includes every condition for which there exists a condition-based consensus protocol. The protocol we have presented in [22] can be instantiated for each particular condition  $C \in \mathcal{C}_f^{wk}$ . It has the same step complexity, whatever the condition it is instantiated with, namely  $O(n \log(f + 1))$  read/write shared memory operations per process.

*Content of the paper.* This paper continues our study of the condition-based approach, from the complexity perspective. It has four main contributions.

1. Although a priori it could be that all conditions of  $\mathcal{C}_f^{wk}$  are equally difficult to solve, it seems plausible that some conditions of  $\mathcal{C}_f^{wk}$  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. In this paper we show that this is indeed the case. For the first contribution we study the structure of the class  $\mathcal{C}_f^{wk}$ , defining a hierarchy of classes of conditions, each one of some degree  $d$  ( $0 \leq d \leq f$ ),

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

where  $\mathcal{C}_f^{[0]} = \mathcal{C}_f^{wk}$  and  $\mathcal{C}_f^{[f]}$  is the class of easiest conditions, also denoted  $\mathcal{C}_f^{st}$ .

2. We present a condition-based consensus protocol that can be used for any condition of degree  $d$ , and shows that the value  $f - d$  upper bounds<sup>1</sup> the “difficulty” of the class  $\mathcal{C}_f^{[d]}$ . More precisely, it is shown that the number of collect operations that are executed by our consensus protocol is related to  $d$ . Roughly speaking, for any condition  $C \in \mathcal{C}_f^{[d]}$ , the number of collect invocations of the wait-free, condition-dependant part of the protocol is proportional to  $\log_2(f - d + 1)$ . Hence, when we progress in the hierarchy from the largest class  $\mathcal{C}_f^{wk} = \mathcal{C}_f^{[0]}$  to the smallest class  $\mathcal{C}_f^{st} = \mathcal{C}_f^{[f]}$ , there are more and more efficient consensus protocols, until one gets  $\mathcal{C}_f^{st}$ , which can be solved with essentially zero collect operations.

The condition-based consensus protocol uses two param-

eters  $P$  and  $S$  (as in [22]) that depend directly on the condition. The predicate  $P$  tells a process if it can decide based on its view of the input vector, and the function  $S$  tells it the value to decide. The 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], is wait-free, and executes a number of collect/write iterations 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 some containment properties. 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. Finally, the last part of the protocol is where a process makes its best effort to terminate. The complexity of a condition is evaluated in terms of the number of steps of the wait-free part of the protocol. The other two parts of the protocol are independent of the condition; the first depends only on  $f$ , while the last does not depend on  $f$  but may never terminate.

3. For each degree  $d$ , two conditions, denoted  $C1_f^{[d]}$  and  $C2_f^{[d]}$ , are presented. These 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], showing that they are of the hardest, i.e., in  $\mathcal{C}_f^{[0]} - \mathcal{C}_f^{[1]}$ .

4. 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 the parameters  $P$  and  $S$  have to satisfy for the above protocol to solve consensus for a condition  $C$  in  $\mathcal{C}_f^{[d]}$ . The second, called *legality*, is in terms of a graph derived from  $C$ ,  $d$  and  $f$ : among its vertices there are the input vectors, and among its edges, there is an edge connecting two input vectors when they differ in at most  $f + d$  entries. The acceptability characterization is useful for deriving consensus protocols, while the legality characterization is more adequate to study noteworthy properties of the hierarchy of conditions. We have the following applications of the characterizations. (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, then every condition of degree  $d$  is trivial (i.e., a single value can be decided); namely, if  $n \leq f + d$ , every condition in  $\mathcal{C}_f^{[d]}$  is trivial. For instance, every condition in  $\mathcal{C}_f^{[f]}$  is trivial when  $f \geq n/2$ . (3) Several relations between the classes of the hierarchy are derived. For instance,  $\mathcal{C}_{2f}^{wk} \subset \mathcal{C}_f^{st}$ , for  $f < n/2$ . This means that (using the protocol complexity  $\log_2(f - d + 1)$ ) for  $C \in \mathcal{C}_{2f}^{wk}$  there is a consensus protocol with complexity  $O(\log(2f + 1))$  tolerating  $2f$  failures, or one with constant complexity tolerating  $f$  failures.

*Theoretical basis and related work.* 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 terminate always, with an output vector satisfying the task specification. Our notion of problem is a kind of “safe task” where, in addition to the requirements of a decision task, processors are

<sup>1</sup>We recently discovered a more efficient protocol for  $f < n/2$ , with only linear complexity [24], showing that our hierarchy is interesting only for  $f \geq n/2$ .

required to satisfy a safety property when inputs 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]). We are able to derive our results using only graph connectivity, due to the simplicity of the allowed output vectors. The main innovation in this context is that the definition of our input graphs depends on the degree  $d$ , and hence its connectivity is affected by a bound on step complexity.

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]<sup>2</sup>. If there turns out to exist a linear time implementation of the snapshot operation, the hierarchy introduced in this paper would collapse.<sup>3</sup> Indeed, in models where linear snapshot implementations are known (e.g., multi-writer registers [19], dynamic test&set or randomized dynamic, single-writer multi-reader [3]) the hierarchy collapses, because one can use such a snapshot implementation (in the algorithm of [22]) to solve consensus for any condition in  $\mathcal{C}_f^{w,k}$  with linear step complexity in the corresponding model.

We remark that the set of acceptable conditions is quite rich. In general, they do not 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.** Section 2 introduces the computation model. Section 3 presents the condition-based approach. Section 4 defines the hierarchy of classes of conditions. Section 5 presents the general condition-based protocol. Section 6 studies the two particular conditions  $C1_f^{[d]}$  and  $C2_f^{[d]}$ . Some proofs are omitted for lack of space, they can be found in [23].

## 2. 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 processes  $p_i$  with an operation  $\text{read}(X[j])$ . Only  $p_i$  can write to the  $i$ -th component,  $X[i]$ , it uses the operation  $\text{write}(v, X[i])$  for this. In addition to the shared memory, each process has a

<sup>2</sup>More precisely, using [20], the proposed protocol can be improved to require  $O(n \log n)$  basic operations per snapshot and  $O(n)$  per update, or vice-versa.

<sup>3</sup>Although this is not the only way of showing that it collapses, as demonstrates our sequel work [24] for  $f < n/2$ .

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  $\forall j : \text{do read}(X[j])$  **enddo**. Hence, it returns an array of values  $[a_1, \dots, a_n]$  such that  $a_j$  is the value returned by  $\text{read}(X[j])$ .

## 3. THE CONDITION-BASED APPROACH FOR CONSENSUS SOLVABILITY

In the *consensus* problem there is a set  $\mathcal{V}$  of values that can be *proposed* by the processes,  $\perp \notin \mathcal{V}$ , and  $|\mathcal{V}| \geq 2$ . 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 denote with  $I$  an input vector with all entries in  $\mathcal{V}$ , and with  $J$  an input vector that may have some entries equal to  $\perp$ . If at most  $f$  processes can crash, we consider only input vectors  $J$  with at most  $f$  entries equal to  $\perp$ , 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  $\perp$ , and 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 consists of considering 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:

- P-Validity: A decided value is a proposed value.
- P-Agreement: No two processes decide different values.
- P-Best\_Effort\_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.

## 4. A HIERARCHY OF CLASSES OF CONDITIONS

This section defines and investigates the hierarchy  $\mathcal{C}_f^{[f]} \subset \mathcal{C}_f^{[f-1]} \subset \dots \subset \mathcal{C}_f^{[1]} \subset \mathcal{C}_f^{[0]}$  of condition classes that allow solving the consensus problem. As previously noted, it was proved in [22] that the largest class in the hierarchy,  $\mathcal{C}_f^{[0]}$ , includes every condition for which a consensus protocol does exist. This study is done in two directions, namely, acceptability and legality of a condition. These notions were introduced in [22] without the *degree* notion. Here they are generalized to any degree  $d$ .

We use the following notation. For vectors  $J1, J2 \in \mathcal{V}_f^n$ ,  $J1 \leq J2$  if  $\forall k : J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$ , and we say that

$J_2$  contains  $J_1$ . Let  $\#_x(J)$  denote the number of entries of  $J$  whose value is  $x$ , with  $x \in \mathcal{V} \cup \{\perp\}$ .

#### 4.1 Acceptability and Legality

Given a condition  $C$  and a value of  $f$ , acceptability is a combinatorial property of  $C$ , inspired on an operational notion defined in terms of a predicate  $P$  and a function  $S$  that have to satisfy some properties in order that a protocol can be designed. Those properties are related to termination, validity and agreement, respectively.

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

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

The second property is related to validity.

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

The next property concerns agreement. Given an input vector  $I$ , if two processes  $p_i$  and  $p_j$  get the views  $J_1$  and  $J_2$ , and both belong to  $\mathcal{I}_f$  such that  $P(J_1)$  and  $P(J_2)$  are satisfied, these processes have to decide the same value of  $\mathcal{V}$ , from  $J_1$  for  $p_i$  and  $J_2$  for  $p_j$ , whenever the following holds, for each integer  $d$  in the range  $0 \leq d \leq f$ .

- Property  $A_{P \rightarrow S}^{[d]}$ :  $\forall I \in \mathcal{V}^n : \forall J_1, J_2 \in \mathcal{I}_f : P(J_1) \wedge P(J_2) \wedge ((J_1 \leq J_2) \vee (\#_{\perp}(J_1) + \#_{\perp}(J_2) \leq f + d)) \Rightarrow S(J_1) = S(J_2)$ .

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

The parameter  $d$  is called the *degree* of the condition. A class of conditions can be analyzed using an alternative representation of its conditions, namely a graph. Given a condition  $C$  and  $0 \leq d \leq f$ , 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,  $J \in \mathcal{I}_f$  for every  $I$  in  $C$ . Two vertices  $I_1, I_2$  of  $C$  are connected by an edge if their Hamming distance  $\text{dist}(I_1, I_2) \leq (f + d)$ , and two views  $J_1, J_2 \in \mathcal{I}_f$  are connected by an edge if  $J_1 \leq J_2$  ( $I$  is connected to all its views  $J \in \mathcal{I}_f$ , since  $I$  belongs to  $\mathcal{I}_f$ ).

**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.** 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 by induction on the edges of the connected component  $C'$  that  $S$  is constant on  $C'$ . We consider two cases representing the two kinds of edges:

- Case of an edge  $(I_1, I_2)$  with  $I_1, I_2 \in C' \cap C$ . By definition of  $G^{[d]}(C, f)$ , let  $x$  be such that  $\text{dist}(I_1, I_2) = x \leq (f + d)$ . Let  $J_1 \in \mathcal{I}_{1f}$  be the view obtained by replacing

the first  $\lceil x/2 \rceil (\leq f)$  entries in which  $I_1$  and  $I_2$  differ by  $\perp$ . Thus,  $P(J_1)$  is true, by Property  $T_{C \rightarrow P}$ . Let  $J_2 \in \mathcal{I}_{2f}$  be the view obtained by replacing the last  $\lfloor x/2 \rfloor (\leq f)$  entries in which  $I_1$  and  $I_2$  differ by  $\perp$ . Thus,  $P(J_2)$  is true, by Property  $T_{C \rightarrow P}$ .

Let  $I$  be the vector obtained by combining all non- $\perp$  entries of  $J_1$  and  $J_2$ . Notice that  $\#_{\perp}(I) = 0$  and  $J_1, J_2 \in \mathcal{I}_f$ , and  $I$  is not necessarily in  $C$ . Since  $J_1, J_2 \in \mathcal{I}_f$ ,  $P(J_1) \wedge P(J_2)$  and  $\#_{\perp}(J_1) + \#_{\perp}(J_2) = x \leq (f + d)$ , Property  $A_{P \rightarrow S}^{[d]}$  implies that  $S(J_1) = S(J_2)$ . Since  $J_1 \in \mathcal{I}_{1f}$  and  $P(J_1) \wedge P(I)$ , Property  $A_{P \rightarrow S}^{[d]}$  gives  $S(I) = S(J_1)$ . Similarly, we have  $S(I) = S(J_2)$ . Consequently,  $S(I) = S(J_2)$ .

- Case of an edge  $(J_1, J_2)$  with  $J_1, J_2 \in \mathcal{I}_f$  and  $J_1 \leq J_2$ . Property  $A_{P \rightarrow S}^{[d]}$  and Property  $T_{C \rightarrow P}$  imply that  $S(J_1) = S(J_2)$ .

$\Leftarrow$  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  $J_1, J_2 \in \mathcal{I}_f$ , such that  $P(J_1) \wedge P(J_2)$  hold. We consider the two cases of the definition of  $A_{P \rightarrow S}^{[d]}$ :

- Case  $J_1 \leq J_2$ .  $J_1$  and  $J_2$  are connected in  $G^{[d]}(C, f)$ . Hence,  $S(J_1) = S(J_2)$  by definition of  $S$ .

- Case  $\#_{\perp}(J_1) + \#_{\perp}(J_2) \leq (f + d)$ . We need to show that  $S(J_1) = S(J_2)$ . Since  $P(J_1) \wedge P(J_2)$  holds, by definition of  $P$ , there exist  $I_1, I_2$  in  $C$ , with  $J_1 \in \mathcal{I}_{1f}$  and  $J_2 \in \mathcal{I}_{2f}$ . Thus,  $\text{dist}(I_1, I) = \#_{\perp}(J_1)$  and  $\text{dist}(I, I_2) \leq \#_{\perp}(J_2)$ , and hence  $\text{dist}(I_1, I_2) \leq (\#_{\perp}(J_1) + \#_{\perp}(J_2)) \leq (f + d)$ . That is,  $I_1, I_2$  are vertices of  $G^{[d]}(C, f)$  joined by an edge, and hence with  $S(I_1) = S(I_2)$ . Similarly,  $I_1, J_1$  and  $I_2, J_2$  are vertices of  $G^{[d]}(C, f)$  joined by an edge, and  $S(I_1) = S(J_1)$ ,  $S(I_2) = S(J_2)$ . The equality  $S(J_1) = S(J_2)$  follows and terminates the proof of the theorem.  $\square$  *Theorem 1*

Let us notice that if  $C$  is  $(f, d)$ -acceptable (or equivalently by the previous theorem  $(f, d)$ -legal) then  $C$  is  $(f, d - 1)$ -acceptable (i.e.,  $(f, d - 1)$ -legal). 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  $G^{[d-1]}(C, f)$ . In the next section we will use this property to define the hierarchy.

The two extremes,  $d = f$  and  $d = 0$ , are particularly interesting. We will use *f-weak-acceptability* as a synonym of  $(f, 0)$ -acceptability (or  $(f, 0)$ -legality), and *f-strong-acceptability* as a synonym of  $(f, f)$ -acceptability (or  $(f, f)$ -legality). Then it is easy to check that we can use

- Property  $A_{P \rightarrow S}^{st}$ :  $\forall I \in \mathcal{V}^n : \forall J_1, J_2 \in \mathcal{I}_f : P(J_1) \wedge P(J_2) \Rightarrow S(J_1) = S(J_2)$ ,

instead of  $A_{P \rightarrow S}^{[f]}$  in the definition of  $(f, f)$ -acceptability. Also, we can use

- Property  $A_{P \rightarrow S}^{wk}$ :  $\forall I \in \mathcal{V}^n : \forall J_1, J_2 \in \mathcal{I}_f : (J_1 \leq J_2) \wedge P(J_1) \wedge P(J_2) \Rightarrow S(J_1) = S(J_2)$ ,

instead of  $A_{P \rightarrow S}^{[0]}$  in the definition of  $(f, 0)$ -acceptability, although this is not as easy to see [22].

## 4.2 The Hierarchy

Here we describe the hierarchy of conditions that allow solving 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).*

The next theorem provides the complete hierarchy of classes of conditions. We already discussed the reason for the  $\subseteq$  containments. We shall later prove, in Theorem 9, that they are strict.

**THEOREM 2.**

$$(\mathcal{C}_f^{st} =) \mathcal{C}_f^{[f]} \subset \mathcal{C}_f^{[f-1]} \subset \mathcal{C}_f^{[f-2]} \subset \dots \subset \mathcal{C}_f^{[0]} (= \mathcal{C}_f^{wk}).$$

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  $d$ .<sup>4</sup>

**THEOREM 3.** *The class  $\mathcal{C}_f^{[d]}$  is decidable in polynomial time, for all  $d, 0 \leq d \leq f$ .*

**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  $\mathcal{C}_f^{st}$ , and hence to all other classes. This motivates the following definition

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

<sup>4</sup>However, one could envision very compact representations of  $C$ , such that the algorithm might not be polynomial in the size of that representation.

The following theorem, a direct consequence of the legality notion, says that the class  $\mathcal{C}_f^{[d]}$  is interesting only when  $f < n - d$ . When considering the class of the strongest conditions ( $d = f$ ), this means the class  $\mathcal{C}_f^{[f]}$  is trivial when  $f < n/2$ .

**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  $(P, S)$  that make  $C$   $(f, d)$ -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 “richness” of a condition. For example, for  $f < n/2$ ,  $\mathcal{C}_{2f}^{wk} \subseteq \mathcal{C}_f^{st}$ . More generally, this is expressed by the following “trading” theorem:

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

**Proof** 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  $C \in \mathcal{C}_f^{[d]}$ .  $\square_{\text{Theorem 5}}$

## 4.3 Defining Predicates $P$ for the Classes $\mathcal{C}_f^{wk}$ and $\mathcal{C}_f^{st}$

We conclude from Theorem 5 that  $\mathcal{C}_{2f}^{wk} = \mathcal{C}_{2f}^{[0]} \subseteq \mathcal{C}_f^{[f]} = \mathcal{C}_f^{st}$ . But, if a  $2f$ -weak-acceptable condition is given with a pair of associated parameters  $(P, S)$ , this theorem does not provide a systematic way to derive a pair of parameters for the corresponding  $f$ -strong-acceptable condition. The theorem that follows provides such a systematic construction.

**THEOREM 6.** *Let  $f < n/2$ . If  $C$  is a  $2f$ -weak-acceptable condition with parameters  $(P, S)$ , then  $C$  is  $f$ -strong acceptable with parameters  $(P_{min}, S)$ , where  $P_{min}(J) \equiv (\exists I \in C : J \in \mathcal{I}_f)$ .*

**Proof** Let  $C$  be a  $2f$ -weak-acceptable condition with parameters  $(P, S)$ .  $C$  is  $(2f, 0)$ -acceptable, or equivalently  $(2f, 0)$ -legal. From Theorem 5, it follows that  $C$  is  $(f, f)$ -legal, which is the definition of the  $f$ -strong-legality. As  $\mathcal{I}_f \subset \mathcal{I}_{2f}$ , it follows that  $S$  is well-defined on  $\mathcal{I}_f$ . Moreover,  $P_{min}$  trivially satisfies  $\text{TC}_{C \rightarrow P_{min}}$ . It follows that  $C$  is  $f$ -strong acceptable with  $(P_{min}, S)$ .  $\square_{\text{Theorem 6}}$

Given an  $f$ -strong-acceptable condition  $C$  with parameters  $P$  and  $S$ , the following theorem introduces a systematic way to associate with  $C$  a pair of predicates  $(P', S')$  such that  $P'$  may be better than  $P$  in the following sense:  $\forall J \in \mathcal{V}_f^n : P(J) \Rightarrow P'(J)$ .

**THEOREM 7.** *Let  $C$  be an  $f$ -strong-acceptable condition (i.e.,  $C \in \mathcal{C}_f^{[f]}$ ) with associated parameters  $(P, S)$ .  $C$  is  $f$ -strong-acceptable with the parameters  $(P', S')$  defined as follows:*

- $P'(J) \equiv (\exists J_0 \leq J : (\#_{\perp}(J_0) \leq f) \wedge P(J_0))$ ,
- $S'(J) = S(J_0)$  where  $J_0$  is such that  $(J_0 \leq J) \wedge (\#_{\perp}(J_0) \leq f) \wedge P(J_0)$ .

**Proof** See [23].

□<sub>Theorem 7</sub>

## 5. A CONDITION-BASED CONSENSUS PROTOCOL FOR $\mathcal{C}_F^{[d]}$

The *Consensus* protocol presented in Figure 1 is an  $n$  process protocol that solves the consensus problem for any condition  $C$  of degree  $d$  and  $f$ , once its parameters  $P, S$  have been correspondingly instantiated.<sup>5</sup> It generalizes the protocol we presented in [22] (that works only for  $d = 0$ ) by including a `strong_collect` procedure that partially orders views. 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]), and the correctness proofs are similar.

### 5.1 The Consensus Protocol

To take into account and exploit the degree  $d$  of a condition, the protocol uses a `strong_collect` abstraction. We first define its specification, and based on it, prove the correctness of the consensus protocol. The next subsection presents an implementation of the `strong_collect` abstraction

*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 the number of  $\perp$  values exceed some threshold. 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, ordering by containment *all* views, a snapshot 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 reduces, 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 1.** `strong_collectf[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  $J$  be the union of all  $J_i$  vectors. Then, they eventually get back views  $\{J'_i\}$  resp., such that:

1.  $J_i \leq J'_i \leq J$ ,

<sup>5</sup>As we have seen, the class  $\mathcal{C}_f^{[d]}$  is interesting only when  $f < n - d$ . Moreover, in our sequel [24] there is a protocol with no collect in the wait-free part when  $f < n/2$ . Hence, this section is interesting for  $f \geq n/2 > d$ . Which is exactly the case where there is no message passing solution [22].

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 steps executed by a process  $p_i$  is:  $O(n \log(f - d + 1))$ .

*The consensus protocol.* The protocol assumes  $P$  and  $S$  have been instantiated to correspond to a condition  $C$  that belongs to the class  $\mathcal{C}_f^{[d]}$ , so that  $P, S$  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 `Consensusf[d](vi)` 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 protocol has the three-part structure described in the Introduction:

**Part 1** (lines 1-2): A process  $p_i$  first writes its input value  $v_i$  to a 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** (lines 3-6): Now,  $p_i$  enters its wait-free, condition-dependent part of the protocol. It uses the `strong_collectf[d]` underlying abstraction to compute a possibly enriched view  $J'_i$  according to Specification 1. With the view  $J'_i$ , it tries to make a decision, by evaluating  $P(J'_i)$ . If true,  $p_i$  returns  $S(J'_i) = w_i$  (line 6), otherwise (i.e.,  $w_i = \top$ )  $p_i$  proceeds to the next part, the best effort termination section. 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** (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 proceed to decide according to a fixed, deterministic rule  $F$  that returns one of the input values (such as `max`).

**Function** `Consensusf[d](vi)`:

```

(1) write(vi, V[i]);
(2) repeat Ji ← collect(V) until (#⊥(Ji) ≤ f);
(3) J'i ← strong_collectf[d](Ji);
(4) if P(J'i) then wi ← S(J'i) else wi ← ⊤;
(5) write(wi, W[i]);
(6) if (wi ≠ ⊤) then return(wi)
(7)           else repeat Xi ← collect(W);
(8)                   if (∃ j : Xi[j] ≠ ⊥, ⊤)
(9)                     then return(Xi[j])
(10)                  until (⊥ ∉ Xi);
(11)                  [a1, . . . , an] ← collect(V);
(11)                  return(F([a1, . . . , an]))

```

**Figure 1: Degree  $d$  Consensus Protocol**

Recall that the step complexity considers only the second, wait-free part of the protocol, that is, the shared memory operations performed by the `strong_collect` subprotocol. The other parts are independent of  $d$  and  $C$ .

**THEOREM 8.**  $\text{Consensus}_f^{[d]}$  solves the consensus problem for  $f$  and any condition  $C \in \mathcal{C}_f^{[d]}$  if its parameters  $P, S$  satisfy Property  $\text{TC}_{\rightarrow P}$ , Property  $\text{VP}_{\rightarrow S}$ , and Property  $\text{A}_{P \rightarrow S}^{[d]}$ . The step complexity of a process is  $O(n \log(f - d + 1))$ .

**Proof** P-Validity: It follows from the code, from Property  $\text{VP}_{\rightarrow S}$ , and from the first item of Specification 1, that a decided value is a proposed value.

P-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  $\text{A}_{P \rightarrow S}^{[d]}$ . If  $\#_{\perp}(J'_i) + \#_{\perp}(J'_j) > f + d$ , then, due to the second item of Specification 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  $\text{A}_{P \rightarrow S}^{[d]}$ .

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.

P-Best\_Effort\_Termination: The proof is similar to the one in [22], using Specification 1 and Property  $\text{TC}_{\rightarrow P}$ .

Step Complexity: Follows directly from item 3 of Specification 1.  $\square_{\text{Theorem 8}}$

## 5.2 Implementing the strong\_collect Abstraction

This section describes the implementation of the strong\_collect abstraction in Figure 2. 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 son of the vertex, and the other views proceed to the left son. 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 less  $\perp$  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 1.

**Traversing a labeled binary tree.** The tree used by strong\_collect is a data structure shared by the processes which 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 initialized to  $[\perp, \dots, \perp]$  that can atomically be written by  $p_i$  (to store its current view) and read by any process. Each vertex  $v$  is labeled with a pair of integers, namely,  $L(v)$  and  $H(v)$  defining an integer interval associated with  $v$ .

Let the size of a view  $J$  be  $|J| = n - \#_{\perp}(J)$  (number of positions of  $J$  with a non- $\perp$  value). As suggested before,

each process  $p_i$  traverses the tree from the root downwards modifying its initial view and getting a possibly enriched view such that its size is always in the interval  $[L(v), H(v)]$  of the vertex  $v$  being traversed. An interval is *trivial* if its size is 1. The processes that terminate in a leaf  $v$  with a trivial interval, namely,  $L(v) = H(v) = x$ , get the same view (of size  $x$ ), while the views of processes that terminate in leaves with non-trivial intervals are not guaranteed to be ordered by containment. In addition, the tree structure and its traversal guarantee that any view  $J$  dominates any view  $J'$  obtained in a leaf on the left of that view, i.e.,  $J' \leq J$ . Any tree can be used, as long as the intervals of the children of a vertex form a partition of the parent interval, and the interval of the root contains the sizes of the initial views. More precisely, the labels are set to satisfy the following properties (hence, the intervals associated with the leaves of the tree form a partition of the interval associated with the root):

- $[L(root), H(root)] = [n - f, n]$ ,
- $L(v.left) = L(v)$ ,  $H(v.left) = L(v.right) - 1$  and  $H(v.right) = H(v)$  for any non-leaf vertex  $v$ .

When a process  $p_i$  calls  $\text{strong\_collect}_f^{[d]}(J_i)$ , we have  $J_i \in \mathcal{I}_f$ , where  $I$  is the input vector associated with the current execution. The process starts at the root with its initial view, namely,  $current_i = J_i$ . Then, it uses an underlying abstraction  $\text{classifier\_improver}$  (see below) to ameliorate its view. More precisely, at line 4 the process calls  $root.classifier\_improver(current_i)$  (which uses  $H(root.left)$  as the boundary between the intervals of the children of  $root$ , and  $root.R$  as shared array), and gets back a “new” view stored in  $current_i$ , and a boolean value ( $rich_i$ ) indicating if the view is rich for this node (for the root of the tree,  $rich_i$  is true if the view has more than  $H(root.left)$  entries different from  $\perp$ ). If so, the process moves to the right son of  $root$ , while if the view is not rich enough, it moves to its left son. This is repeated until a leaf of the tree is reached.

**Function**  $\text{strong\_collect}_f^{[d]}(J_i)$ :  $result(J'_i)$

- (1)  $current_i \leftarrow J_i$ ;
- (2)  $v \leftarrow root$ ;
- (3) **while**  $v$  is not a leaf **do**
- (4)  $(current_i, rich_i) \leftarrow v.classifier\_improver(current_i)$ ;
- (5) **if**  $rich_i = \text{yes}$  **then**  $v \leftarrow v.right$  **else**  $v \leftarrow v.left$
- (6) **endwhile**;
- (7) **return**  $(J'_i = current_i)$

**Figure 2: An Implementation of  $\text{strong\_collect}_f^{[d]}$**

**The classifier\_improver abstraction.** The  $v.classifier\_improver$  abstraction is attached to the vertex  $v$  of the tree. The notation  $\cup\{J_1, \dots, J_n\}$ , for views  $J_i \in \mathcal{I}_f$  denotes the view  $J$  that has a non- $\perp$  value  $v = J[j]$  in a position  $j$  if at least for one  $i$ ,  $J_i[j] = v$ . A classifier\_improver protocol should satisfy the following specification.

**SPECIFICATION 2.**  $v.classifier\_improver$  is an  $n$  process wait-free protocol attached to the vertex  $v$ , with associated integer interval  $[L(v), H(v)]$ . Processes  $p_i$  invoke it with views  $J_i$  such that  $J_i \in \mathcal{I}_f$ , for some input vector  $I$ , and  $|\cup\{J_i\}| \leq$

$H(v)$ . They eventually get back views  $J'_i \in \mathcal{I}_f$  and booleans  $rich_i$ , such that:

1.  $J_i \leq J'_i$ ,
2. If  $rich_i = \text{yes}$  then  $|J'_i| > H(v.\text{left})$  else  $|J'_i| \leq H(v.\text{left})$ ,
3.  $(rich_i = \text{no} \wedge rich_j = \text{yes}) \Rightarrow (J'_i < J'_j)$ ,
4.  $\cup\{J'_i\} = \cup\{J_i\}$ ,
5.  $|\cup\{J'_i \text{ s.t. } rich_i = \text{no}\}| \leq H(v.\text{left})$ .

The execution of  $v.\text{classifier\_improver}$  by  $p_i$  (line 4), during a  $\text{strong\_collect}$  invocation, is denoted  $p_i \in \text{visited}(v)$ . Moreover,  $J_{i,v}$  denotes the actual value of the corresponding input parameter ( $\text{current}_i$ ), while  $rich_{i,v}$  denotes the value returned by  $\text{classifier\_improver}$  at the end of the visit of  $v$  by  $p_i$ . The lemma that follows states the main properties of the traversal of an arbitrary tree  $T$ .

LEMMA 1. 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.  $|\cup_{p_i \in \text{visited}(v)} \{J_{i,v}\}| \leq H(v)$ ,
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** See [23].  $\square_{\text{Lemma 1}}$

*Designing an appropriate tree.* We now design a tree  $T^{[d]}$  for each  $d$  in  $[0, f]$ , to be used by the protocol  $\text{strong\_collect}_f^{[d]}$ . We assume  $d < f$ , because otherwise the protocol is not needed: no views need to be ordered, and no tree is needed. To satisfy Specification 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 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  $T^{[d]}$  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  $T^{[d]}$  previously defined.

LEMMA 2. The  $\text{strong\_collect}_f^{[d]}$  protocol described in Figure 2 satisfies Specification 1.

**Proof** The proof of Specification 1(1) is a direct consequence of the classifier\_improver Specification 2(1).

To prove Specification 1(2), we consider two returned views  $J'_i, J'_j$  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 1(3) or else they are equal by Lemma 1(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 1(3).

The proof of Specification 1(3) follows from (1) the height of  $T^{[d]}$  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.  $\square_{\text{Lemma 2}}$

COROLLARY 1. The step complexity of the strong\_collect protocol (and hence the step complexity of the wait-free part of the Consensus $_f^{[d]}$  protocol) is at most

$$(2n + 1) \left\lceil \log_2 \left( \left\lceil \frac{f - d}{2} \right\rceil + 1 \right) \right\rceil.$$

**Proof** Immediate consequence of the height of the tree  $T^{[d]}$ , and the fact that a collect costs  $n$  steps.  $\square_{\text{Corollary 1}}$

### 5.3 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 Figure 3. It works as follows. The processes write their views  $J_i$  to a shared array  $v.R$ , and end up with views  $J'_i$  either of size 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.

**Function**  $v.\text{classifier\_improver}(J_i)$ :  $\text{result}(J'_i, rich_i)$

- (1)  $\text{write}(J_i, v.R[i]);$
- (2)  $R_i \leftarrow \text{collect}(v.R);$
- (3) **if**  $|\cup\{R_i[1], \dots, R_i[n]\}| > H(v.\text{left})$
- (4)     **then**  $R_i \leftarrow \text{collect}(v.R);$
- (5)     **return**  $( J'_i = \cup\{R_i[1], \dots, R_i[n]\},$   
 $rich_i = \text{yes}$ )
- (6)     **else return**  $( J'_i = J_i,$   
 $rich_i = \text{no}$  )

Figure 3: An Implementation of classifier\_improver

LEMMA 3. Let us consider a vertex  $v$ . The  $v.\text{classifier\_improver}$  protocol described in Figure 3 satisfies Specification 2.

**Proof** The proof of Specification 2(1) follows directly from the code and the fact that,  $\forall J_i, \dots, J_j$ , we have  $J_i \leq J_i \cup \dots \cup J_j$ .

The proof of Specification 2(2) follows directly from the lines 3-6 (test of line 3 and containment property of sequential collect invocations).

For the proof of Specification 2(3) let us observe that, as  $rich_j = yes$ ,  $p_j$  executes two collect operations, that 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,  $rich_i = no$ ,  $J_i' = J_i$  (from the protocol text). Moreover, as  $rich_i = no$  and  $rich_j = yes$ , we have  $|J_i'| \leq H(v.left) < |J_j'|$  from the protocol test at line 3. Hence,  $J_i' \neq J_j'$ . It follows that  $J_i' < J_j'$ .

- Let us now examine the other case:  $p_i$  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 union 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 2(4), follows directly from the code (no entry value is created by the protocol).

For the proof of Specification 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 union of these inputs must be at most  $H(v.left)$ , since otherwise  $p_i$  would get  $rich_i = yes$  when it executes line 3.  $\square_{Lemma 3}$

## 6. TWO CONDITIONS

Here we present two families of conditions,  $C1_f^{[d]}$ ,  $C2_f^{[d]}$ , for  $0 \leq d \leq f \leq n$ , in the style of the conditions  $C1_f^{wk}$ ,  $C2_f^{wk}$  introduced in [22]. Indeed,  $C1_f^{wk} = C1_f^{[0]}$  and  $C2_f^{wk} = C2_f^{[0]}$ .

### 6.1 The Condition $C1_f^{[d]}$

The idea of this condition is to guarantee that all the processes have the same extremal (largest or smallest) value in their local 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:

$$(I \in C1_f^{[d]}) \text{ iff } [a = \max(I) \Rightarrow \#_a(I) > (f + d)].$$

For ease of notation, in the next theorem we define  $\mathcal{C}_f^{[d]}$  to be empty for  $d = f + 1$ . It follows from Lemma 4, 5, and Theorem 1.

**THEOREM 9.** For  $(f + d) < n$ ,  $C1_f^{[d]} \in \mathcal{C}_f^{[d]} - \mathcal{C}_f^{[d+1]}$ .

**LEMMA 4.** If  $(f + d) < n$ ,  $C1_f^{[d]}$  is  $(f, d)$ -acceptable with the following parameters:

- $P1_f^{[d]}(J) \equiv (a = \max(J)) \Rightarrow (\#_a(J) > (f + d - \#_{\perp}(J)))$ ,
- $S1(J) = \max(J)$ .

**Proof** First, we show that  $C1_f^{[d]}$  is  $(f, d)$ -legal, and then exhibit the  $(P, S)$  parameters for its  $(f, d)$ -acceptability.

In order to show that  $C1_f^{[d]}$  is  $(f, d)$ -legal, we prove that  $S1 \equiv \max$  is constant on any connected component of  $G^{[d]}(C1_f^{[d]}, f)$ . Let  $C'$  be a connected component of  $G^{[d]}(C1_f^{[d]}, f)$ . We consider two cases, induced by the two types of vertices:

- Let  $I1, I2$  be two vertices of  $C1_f^{[d]}$ , connected in  $C'$ . Let  $a$  (resp.  $b$ ) be the maximum of  $I1$  (resp.  $I2$ ). Assume that  $a \geq b$ . We have  $\#_a(I2) \geq \#_a(I1) - d(I1, I2) \geq \#_a(I1) - (f + d)$ , because  $I1$  and  $I2$  are connected by an edge. Since  $I1$  belongs to  $C1_f^{[d]}$ , it satisfies  $\#_a(I1) > (f + d)$ , and hence  $\#_a(I2) > 0$ , i.e.,  $a$  belongs to  $I2$ . Finally, as  $a \geq b$ , we can conclude that  $a = \max(I2)$ , i.e.,  $S1(I1) = S1(I2)$ .
- Let  $J1, J2$  be two vertices of  $C1_f^{[d]}$ , such that  $J2 \leq J1$  (hence, there is an edge connecting them in  $C'$ ). Let  $I$  in  $C1_f^{[d]} \cap C'$  such that  $J1 \in \mathcal{I}_f$ , and  $a$  the maximum of  $I$ . Since  $I$  belongs to the condition,  $a$  appears at least  $(f + d + 1)$  times in  $I$ . As  $J1$  is obtained by replacing up to  $f$  entries of  $I$  by  $\perp$ , we can conclude that  $a \in J1$ . As the same applies to  $J2$ , we get  $S1(J1) = S1(J2) = a$ .

It follows that  $S1$  is constant in each connected component  $C'$  of  $G^{[d]}(C1_f^{[d]}, f)$ . Consequently,  $C1_f^{[d]}$  is  $(f, d)$ -legal, and due to Theorem 1,  $(f, d)$ -acceptable.

Let us consider the pair  $(P, S)$  used in the proof of the Theorem 1

- $P(J) \equiv (\exists I \in C1_f^{[d]} : J \in \mathcal{I}_f)$ ,
- $S(J) =$  a non- $\perp$  value common to the vertices of the connected component to which  $J$  belongs.

$S1$  is  $S$  instantiated with  $\max$  (which is constant on a connected component). We show that  $P1_f^{[d]}$  is exactly  $P$ .

Let us first prove  $\forall J : P1_f^{[d]}(J) \Rightarrow P(J)$ . Let  $J$  such that  $P1_f^{[d]}(J)$  holds, i.e.,  $(\max(J) = a) \Rightarrow \#_a(J) > (f + d - \#_{\perp}(J))$ . Let  $I$  be the vector obtained by replacing each  $\perp$  in  $J$  by  $a$ . So,  $J \in \mathcal{I}_f$ . Trivially,  $\#_a(I) > (f + d)$ , and  $I \in C1_f^{[d]}$ . Hence,  $P(J)$  holds.

Let us now prove  $\forall J : P(J) \Rightarrow P1_f^{[d]}(J)$ . Let  $J$  such that  $P(J)$  holds. Let  $I \in C1_f^{[d]}$  such that  $J \in \mathcal{I}_f$ , and  $a = \max(I)$ . Notice that  $\#_a(I) > f + d$ . Combined with  $J \leq I$ , we get  $\#_a(J) > (f + d - \#_{\perp}(J)) \geq 0$ . Hence, the value  $a$  belongs to the vertex  $J$ . As, except for  $\perp$ , the values of  $J$  belong to  $I$ , we have  $a = \max(J)$ . Consequently,  $P1_f^{[d]}(J)$  holds.  $\square_{Lemma 4}$

Using  $C1_f^{[d]}$ , the following lemma proves that there is a  $(f, d)$ -legal condition that is not  $(f, d + 1)$ -legal. A forward reference to this lemma appeared in Theorem 2 to prove strict class containment.

**LEMMA 5.** Assume  $(f + d) < n$  and  $d < f$ . Then,  $C1_f^{[d]}$  is not  $(f, d + 1)$ -legal.

**Proof** Assume for contradiction that  $(f + d) < n$  and  $C1_f^{[d]}$  is  $(f, d + 1)$ -legal. Consider its graph  $G^{[d+1]}(f, C1_f^{[d]})$ . Consider the following vertices  $I1, I2 \in C1_f^{[d]}$ .  $I1$  has the first

$(f + d + 1)$  positions equal to a value  $a$ , and the others equal to  $b$ , such that  $a > b$ , and  $I_2$  has all its entries equal to  $b$ . Clearly,  $I_1$  is in the same connected component that the vector with all entries equal to  $a$ , since we can switch one by one the  $b$ 's of  $I_1$  to  $a$ 's while maintaining the property that the number of  $a$ 's is more than  $(f + d + 1)$ . Thus, there is one, and only one value in common to all vectors of this component, namely,  $a$ . On the other hand,  $I_2$  has only  $b$ 's, so it is in a different connected component. However,  $dist(I_1, I_2) = (f + d + 1)$ , and hence there is an edge connecting  $I_1$  and  $I_2$  in  $G^{[d+1]}(f, C1_f^{[d]})$ , a contradiction.  $\square_{Lemma 5}$

## 6.2 The Condition $C2_f^{[d]}$

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 notations are required to formally express it:  $\#_{1st}(J)$  denotes the occurrence number of the most common non- $\perp$  value of  $J$ ;  $\#_{2nd}(J)$  denotes the occurrence number of the second most common non- $\perp$  value of  $J$  (if there is no such value,  $\#_{2nd}(J) = 0$ ). With these notations  $C2_f^{[d]}$  can be formally expressed as follows:

$$(I \in C2_f^{[d]}) \text{ iff } [ \#_{1st}(I) - \#_{2nd}(I) > (f + d) ].$$

The following is a consequence of Lemma 6 (that follows) and Theorem 1. The proof of Lemma 6 can be found in [23].

THEOREM 10. For  $(f + d) < n$ ,  $C2_f^{[d]} \in \mathcal{C}_f^{[d]}$ .

LEMMA 6. If  $(f + d) < n$ ,  $C2_f^{[d]}$  is  $(f, d)$ -acceptable with the following parameters:

- $P2_f^{[d]}(J) \equiv \#_{1st}(J) - \#_{2nd}(J) > (f + d - \#_{\perp}(J))$ ,
- $S2(J) = a$  such that  $\#_a(J) = \#_{1st}(J)$ .

## Acknowledgments

We thank Nancy Lynch for discussions that lead to the question considered in this paper.

## 7. REFERENCES

- [1] Afek Y., Attiya H., Dolev D., Gafni E., Merritt M. and Shavit N., Atomic Snapshots of Shared Memory. *Journal of the ACM*, 40(4):873-890, 1993.
- [2] Aguilera M.K. and 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. and Rachman O., Atomic Snapshots Using Lattice Agreement, *Distributed Computing*, 8(3):121-132, 1995.
- [4] Attiya H. and Rachman O., Atomic Snapshots in  $O(n \log n)$  Operations. *SIAM Journal of Computing*, 27(2):319-340, 1998.
- [5] Attiya H. and 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] Garay J. and Berman P., Adaptability and the Usefulness of Hints (Extended Abstract). *6th European Symposium on Algorithms (ESA '98)*, Venice (Italy). Springer-Verlag LNCS #1461, pp. 271-282, 1998.
- [8] Borowsky E., Gafni E., Lynch N.A. and Rajsbaum S., The BG Distributed Simulation Algorithm. To appear in *Distributed Computing*, 2001.
- [9] Biran O., Moran S. and Zaks S., A Combinatorial Characterization of the Distributed 1-Solvable Tasks. *Journal of Algorithms*, 11:420-440, 1990.
- [10] Biran O., Moran S. and 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-Verlag LNCS #484, pp. 206-220, 1990.
- [11] Chandra T. and Toueg S., Unreliable Failure Detectors for Reliable Distributed Systems. *JACM*, 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. and Stockmeyer L., Consensus in the Presence of Partial Synchrony. *JACM*, 35(2):288-323, 1988.
- [14] Dolev D., Lynch N.A., Pinter S., Stark E.W., and Weihl W.E., Reaching Approximate Agreement in the Presence of Faults. *JACM*, 33(3):499-516, 1986.
- [15] Fischer M.J., Lynch N.A. and Paterson M.S., Impossibility of Distributed Consensus with One Faulty Process. *JACM*, 32(2):374-382, 1985.
- [16] Gafni E. and Koutsoupias E., Three-Processor Tasks Are Undecidable. *SIAM Journal of Computing*, 28(3):970-983, 1999.
- [17] Herlihy M.P. and 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. and Rajsbaum S., New Perspectives in Distributed Computing. *Invited Talk, Proc. 24th Int. Symposium on Mathematical Foundations of Computer Science (MFCS'99)*, Springer-Verlag LNCS #1672, pp. 170-186, 1999.
- [19] Inoue M., Chen W., Masuzawa T., and Tokura N., Linear-Time Snapshot Using Multi-Writer Multi-Reader Registers. *Proc. 8th Int. Workshop on Distributed Algorithms (WDAG'94)*, Springer-Verlag LNCS #857, pp. 130-140, October 1994.
- [20] Israeli A., Shaham A., and Shirazi A., Linear-Time Snapshot Protocols for Unbalanced Systems, *Proc. 7th Int. Workshop on Distributed Algorithms*, Springer-Verlag LNCS #725, pp. 26-38, 1993.
- [21] Lynch N.A., *Distributed Algorithms*. Morgan Kaufmann Pub., San Francisco (CA), 872 pages, 1996.
- [22] Mostefaoui A., Rajsbaum S. and 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, Crete (Greece), July 2001.
- [23] Mostefaoui A., Rajsbaum S., Raynal M., and Roy M., A Hierarchy of Conditions for Consensus Solvability. *Research Report #1381*, IRISA, University of Rennes, France, January 2001. Available at: [www.irisa.fr/bibli/publi/pi/2001/1381/1381.html](http://www.irisa.fr/bibli/publi/pi/2001/1381/1381.html)
- [24] Mostefaoui A., Rajsbaum S., Raynal M. and Roy M., Condition-Based Protocols for Set Agreement Problems. *Research Report #1393*, IRISA, University of Rennes, France, April 2001, 21 pages. <http://www.irisa.fr/bibli/publi/pi/2001/1393/1393.html>.
- [25] Mostefaoui A., Raynal M. and 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, June 2000.