Conditions on Input Vectors for Consensus Solvability in Asynchronous Distributed Systems

ACHOUR MOSTEFAOUI

Irisa/Ifsic, Université de Rennes, France

SERGIO RAJSBAUM

Instituto de Matemáticas, UNAM, Mexico

AND

MICHEL RAYNAL

Irisa/Ifsic, Université de Rennes, France

Abstract. This article introduces and explores the *condition-based* approach to solve the consensus problem in asynchronous systems. The approach studies *conditions* that identify sets of input vectors for which it is possible to solve consensus despite the occurrence of up to f process crashes. The first main result defines *acceptable* conditions and shows that these are exactly the conditions for which a consensus protocol exists. Two examples of realistic acceptable conditions are presented, and proved to be maximal, in the sense that they cannot be extended and remain acceptable. The second main result is a generic consensus shared-memory protocol for any acceptable condition. The protocol always guarantees agreement and validity, and terminates (at least) when the inputs satisfy the condition with which the protocol has been instantiated, or when there are no crashes. An efficient version of the protocol is then designed for the message passing model that works when $f < n/2$, and it is shown that no such protocol exists when $f \geq n/2$. It is also shown how the protocol's safety can be traded for its liveness.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed systems—*distributed applications*; *network operating systems*; C.4 [**Performance of Systems**]: *fault tolerance*; *reliability, availability, and serviceability*; D.1.3 [**Programming Techniques**]: Concurrent Programming—*distributed programming*; D.4.5 [**Operating Systems**]: Reliability—*fault-tolerance*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*relations*

A preliminary version of this article appeared in *Proceedings of the 33rd ACM Symposium on Theory of Computing* (Crete, Greece, July). ACM, New York, 2001, pp. 153–162.

Part of this work was done while S. Rajsbaum was at HP Research Lab, One Cambridge Center, Cambridge, MA 02139.

Authors' addresses: A. Mostefaoui and M. Raynal, IRISA, Campus de Beaulieu, Universite de Rennes 1, Avenue du General Leclerc, 35042 Rennes Cedex, France, e-mail: raynal@irisa.fr; S. Rajsbaum, Instituto de Matemáticas, UNAM, Mexico.

between models; *computability theory*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism and concurrency*; *alternation and nondeterminism*; F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms: Algorithms, Reliability, Theory, Performance

Additional Key Words and Phrases: Asynchronous systems, consensus problem, crash failures, fault-tolerance, message-passing, atomic registers, shared memory

1. *Introduction*

Context of the study. Agreement and coordination problems are crucial for the design of fault-tolerant applications on top of asynchronous distributed systems prone to failures. Among agreement problems, *consensus* is considered fundamental, and many papers have been written on both its practical and its theoretical aspects [Attiya and Welch 1998; Lynch 1996]. It has gained a leadership position as it can be seen as the "greatest common agreement subproblem." Intuitively, this means that particular agreement problems (e.g., atomic broadcast [Chandra and Toueg 1996], shared memory objects [Herlihy 1991]), can be implemented using a solution to the consensus problem. Informally, this problem can be defined in terms of two requirements: each process proposes a value, and each correct process has to (liveness) decide a value such that (safety) there is a single decided value, and the decided value is a proposed value.

Consensus being such an important problem, it is remarkable that it cannot be solved in an asynchronous system where only one process may crash (Fischer et al. [1985] proved the result for message passing systems, and Loui and Abu-Amara [1987] extended it to shared memory systems). Therefore, researchers started investigating ways of circumventing the impossibility result. Two main directions were explored: relaxing the requirements of the consensus problem, and strengthening the assumptions on the system. Of course, most interesting are problems weaker than consensus that are still interesting for applications, and stronger computation models that still reflect realistic distributed systems.

At least two ways of relaxing the consensus requirements have been investigated. An active research area has been on ways of solving the problem using randomization, so that termination is achieved only with high probability (e.g., Ben-Or [1983], or Aumann [1997] for a more recent work, and references herein). Another approach is to require that processes agree with each other only approximately; either processes must eventually decide on real values which are within ε of each other (e.g., Dolev et al. [1986]), or processes can decide on at most k distinct proposed values (e.g., Chaudhuri [1993]).

Also, at least two ways of strengthening the assumptions on the system have been considered. One major research direction consists of adding synchrony assumptions to the system. This is motivated by the fact that real systems often have access to approximately synchronized clocks, and can make use of timeouts to avoid waiting for a message that has been lost, or for a process that has crashed. Partially synchronous systems where delays and relative processor speeds are bounded have been studied in works such as Dolev et al. [1987] and Dwork et al. [1988]. An interesting direction is the *Unreliable Failure Detector* concept [Chandra and Toueg 1996], that abstracts away from the details of how a processor suspects a failure has occurred, without referring to particular synchrony assumptions. This is achieved

by equipping processes with an oracle that provides them with a list of processes suspected to have crashed. Several failure detector-based consensus protocols have been designed (e.g., Chandra and Toueg [1996] and Mostefaoui and Raynal [1999]). The other major research direction consists of assuming that the system includes communication primitives that are stronger than point-to-point message channels or read/write shared registers. A seminal paper in this approach is Herlihy [1991] where it is shown that there are objects that can be used to solve wait-free (tolerating any number of failures) consensus for n processes but not for $n+1$ processes, and that some objects can be used to solve wait-free consensus for any number of processes. For example, while read/write registers cannot be used to solve wait-free consensus even for just two processes, test&set objects can be used to solve wait-free consensus for two, and no more than two, processes.

Some papers try to circumvent the consensus impossibility result combining the two previous approaches, to benefit from the best of "both worlds." Combining failure detection and randomization is explored in Aguilera and Toueg [1998] and Mostefaoui et al. [2000]. Combining relaxation of the termination requirement with stronger assumptions on the system, so that processes rely on "luck" to terminate, is explored in Aspnes [2000].

Results presented in the article. This article introduces and investigates a new approach to tackle the consensus problem. This approach considers the set of possible vectors of values that can be proposed by the processes, and focuses on conditions that identify sets of vectors allowing n processes to solve the consensus problem despite up to f process crashes, in a standard asynchronous model. The intuition that underlies the approach is simple and natural. To illustrate it, let us consider the extreme case where it is a priori known that all the processes propose the same value. Then, consensus is trivially solved (at no cost!), each process deciding the value it proposes. As a less trivial 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 in this case, 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.

More generally, for a given set of input values $\mathcal{V}$, and particular values of n and f, a *condition* is defined to be the set of all vectors over $\mathcal{V}$ that can be proposed by the processes under normal operating conditions. We are interested in protocols that (1) solve consensus when such a condition holds, and (2) are always safe. Safe means that the protocol guarantees agreement (and a decided value is a proposed value), whether the proposed input vector is allowed by the condition or not. In addition, the protocol must terminate in well-behaved scenarios (e.g., failure-free runs) even if the input vector is not in the condition. This is the best we can hope for, since the consensus impossibility result says we cannot require that a consensus protocol terminates always, for every input vector. By guaranteeing that safety is never violated, the hope is that such a protocol should be useful in applications (e.g., Guerraoui and Raynal [2003] and Lamport [1998]).

After having introduced the *condition-based* approach, we present our first main result: a generic condition-based consensus protocol. This protocol uses a predicate P and a function S, that have to be instantiated for each particular condition C. Intuitively, the predicate P tells a process if the input vector could belong to C

(in general, due to failures and asynchrony, the process is able to find out only part of the input vector), and if so, S tells it what value to decide. This protocol has various desirable features. First, it is simple and efficient, and its parameters P and S can be efficiently computed from the condition C. If the condition C it is based on contains the actual input vector and there are at most f crashes then the protocol solves the consensus problem. When the actual input vector does not belong to C the protocol still terminates in many cases, guaranteeing agreement; it terminates when no process crashes, or when one process decides. Thus, the protocol is attractive from both theoretical and practical points of view.

Clearly, our protocol could not possibly work with *every* condition. For example, the condition that includes every input vector reduces the problem to the original consensus problem, which we know by Fischer et al. [1985] that cannot be solved. Our second main result is identifying the class of conditions for which the proposed protocol solves the consensus problem, called *acceptable conditions* and observing that it is an efficiently decidable class. Moreover, we prove that if there is *any* protocol solving the consensus problem for a condition C, then C must be acceptable. Thus, it is somewhat surprising that such a simple and efficient protocol as the one we propose, solves the problem for any condition for which a solution does exist.

Our characterization is in terms of an intermediate notion that we call *condition legality*. This notion is based on a graph defined by the input vectors of the condition, as explained below. Given a condition C (and a value of f and $\mathcal{V}$), we show that the four following assertions are equivalent: ($A1$) C is acceptable, ($A2$) there exists a consensus algorithm for C, ($A3$) there exists an *nonsafe* algorithm that solves consensus provided it only gets inputs belonging to C, ($A4$) C is legal. We prove this by proving the implications $A1 \Rightarrow A2 \Rightarrow A3 \Rightarrow A4 \Rightarrow A1$. The part $A2 \Rightarrow A3$ is trivial, since the requirements for a nonsafe algorithm are weaker than for its safe counterpart. More interestingly, the characterization implies that $A3 \Rightarrow A2$, which means that the safety requirement (2) above does not limit the set of conditions for which a condition-based consensus protocol exists. We present also a direct proof of this fact: we describe how to transform a protocol that solves consensus assuming that only inputs in a condition C are given, to a protocol that also allows inputs outside of C and guarantees the safety requirement.

An intuition for the characterization can be seen through the legality definition. Basically, the input vectors are represented as vertices of a graph, where two input vectors are connected if they differ in at most f entries. Hence, a particular condition C defines disconnected components of this input graph, by eliminating input vectors outside of C. Different conditions define different ways to produce disconnected components. A condition is *legal* if each connected component of its associated graph, has a value that appears in all the input vectors of that component. The intuition is that, according to the particular condition it is supplied with, the generic protocol maps each connected component to an output vector with all entries equal to the same value—a value that occurs in each input vector of the corresponding component.

The protocol is first described in a very simple model (following a methodology advocated in Gafni [1998]): a shared memory model with atomic snapshots. Then, an efficient implementation is discussed for a message passing system with $f < n/2$, that avoids automatic, but less efficient translations such as those of Attiya [2000] and Attiya and Rachman [1998]. It is also shown that no such protocol exists when $f \geq n/2$.

This article also investigates two particular conditions, $C1$ and $C2$. Both are very natural, and are proved to be acceptable. Condition $C1$ is the following one. Given an input vector I, let a be the most often proposed value, and let b be the second most often proposed value. Let $\#_{1st}(I)$ ($\#_{2nd}(I)$) be the number of times a (b) has been proposed. Then, $C1$ accepts all vectors I with $\#_{1st}(I) - \#_{2nd}(I) \geq f + 1$. It is shown that, when $C1$ is satisfied, consensus can be solved by having each process decide the value it sees the most often. Condition $C2$ assumes that the proposed values are ordered and is the following: the largest proposed value v is proposed by at least $(f + 1)$ processes. In this case, all processes can see this value, and can decide on it. Many other conditions can be defined, but $C1$ and $C2$, are in a sense dual and realistic. Moreover, we prove that a slightly refined version of $C1$, and $C2$ are maximal in the sense that any attempt to extend them results in a condition which is not acceptable.

Related work. Following the publication of an extended abstract of this article [Mostefaoui et al. 2001], several works have continued exploring the condition-based approach. In this article we seek to establish the foundations of the condition-based approach, mainly from the computability point of view, and applied to the consensus problem in asynchronous systems. The case of synchronous systems is explored in Mostefaoui et al. [2003] and Zibin [2003]. All the following works are for asynchronous systems. Efficiency aspects of the approach are studied in a companion paper [Mostefaoui et al. 2001b] where it is shown that the acceptable conditions form a hierarchy, more precisely, the efficiency of the consensus protocols depends of the position of the condition in the hierarchy. In Attiya and Avidor [2002] and Mostefaoui et al. [2002] the condition-based approach is applied to k-set agreement problems. In this article, we give two natural examples of conditions for consensus, $C1$ and $C2$. Other conditions are explored in Mostefaoui et al. [2001a]. A formulation of conditions in terms of error correcting codes has been proposed in Friedman et al. [2002].

The idea of considering restricted set of inputs to a problem is very natural and has appeared in various contexts; just to mention a few examples, it has appeared in on-line algorithms [Azar et al. 1993], adaptive sorting [Castro and Wood 1992], etc. Agreement problems with a restricted set of inputs vectors were considered in Taubenfeld et al. [1994] and Taubenfeld and Moran [1996], where possibility and impossibility results in a shared memory system and a hierarchy of problems that can be solved with up to f failures but not for $(f + 1)$ failures are introduced. More generally, an approach for designing algorithms in situations where there is some information about the typical conditions that are encountered when the respective problem is solved is presented in Berman and Garay [1998]. In this article, the consensus problem in a synchronous setting is analyzed taking as parameter the difference between the number of 0's and 1's in the input vector.

The foundation underlying the proposed condition-based approach can be formalized using topology (e.g., Herlihy and Rajsbaum [1999]). 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. We can call our notion of problem *safe task*, where in addition to the requirements of a decision task, processors are required to satisfy a safety property when inputs are illegal, without necessarily terminating. From this point of view, our article is a study of the class of all safe tasks, with a particular kind of output

vectors: all decisions are equal. Thus, our result is an efficiently decidable characterization of the f-fault tolerant solvability of these safe tasks. In Section 3.2, we explore the relation between safe tasks and decision tasks, and show that our notion of safe task is equivalent (for consensus solvability) to a common notion of decision task.

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 [Biran et al. 1990a]), and leads to undecidable characterizations [Gafni and Koutsoupias 1999; Herlihy and Rajsbaum 1997] (NP-Hard for $f = 1$ [Biran et al. 1990b]). We are able to derive an efficiently decidable characterization of the acceptable conditions (and hence of solvability of consensus safe tasks) using only graph connectivity, due to the simplicity of the allowed output vectors (all entries are equal). For the necessary part of the characterization we use ideas introduced in Biran et al. [1990a] and Moran and Wolfstahl [1987] for $f = 1$, and apply them for any f.

Organization of the article. The paper is made up of 10 sections. After this introduction, Section 2 introduces the computation model, and the consensus problem. Section 3 presents the condition-based approach. Section 4 defines the generic condition-based consensus protocol, proving that it works for all acceptable conditions. Section 5 shows the other direction of this claim, providing a characterization of the conditions allowing to solve the consensus problem. Section 6 studies the conditions $C1$ and $C2$. Section 7 shows the maximality of $C1'$ (a refined version of $C1$) and $C2$. Section 8 explains the adaptation to the message passing model. Section 9 describes the possibility of trading safety for liveness. Finally, Section 10 concludes the article.

2. *Preliminaries*

2.1. THE MODEL. For most of the article we consider the usual asynchronous shared-memory system with n, $n > 1$, processes $p_1, \ldots, p_n$, where at most f, $0 \leq f < n$, processes can crash. The shared memory consists of single-writer, multi-reader atomic registers. The executions are assumed to be linearizable [Herlihy and Wing 1990]. For details of this model, see any standard textbook such as Attiya and Welch [1998] and Lynch [1996]. In Section 8, we extend some of our results to a message-passing model.

The shared memory is organized into arrays. The jth entry of an array $X[1 \cdots n]$ can be read by any process p_i with an operation read($X[j]$). Only p_i can write to the ith component, $X[i]$, and it uses the operation write(v, $X[i]$) when it wants to write value v. To simplify the description of our algorithms, we assume that processes can take atomic snapshots of any of the shared arrays: with snapshot(X) a process p_j atomically reads the content of all the registers of the array X. This assumption is made without loss of generality, since it is known [Afek et al. 1993] that atomic snapshots can be wait-free implemented from single-writer multi-reader registers (although there is a cost in terms of efficiency: the best known simulation has $O(n \log n)$ complexity [Attiya and Rachman 1998]).

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

2.2. THE CONSENSUS PROBLEM. The classic consensus problem has been informally stated in the introduction: every correct process p_i *proposes* a value v_i

and all correct processes have to *decide* on the same value v, that has to be one of the proposed values. More precisely, there is a set $\mathcal{V}$ of values that can be proposed by the processes, $\perp \notin \mathcal{V}$, and $|\mathcal{V}| \geq 2$. Each process starts an execution with an arbitrary input value from $\mathcal{V}$, the value it proposes, and has to irrevocably decide some value. We say *an f-fault tolerant protocol solves the consensus problem* if all its executions satisfy the following properties:

—Validity. A decided value is a proposed value.

—Agreement. No two processes decide differently.[1]

—Termination. If no more than f processes crash, each correct process eventually decides some value.

A consensus problem is *binary* when $|\mathcal{V}| = 2$. Otherwise, it is *multivalued*. The set $\mathcal{V}$ may be a priori known by the processes or not.

3. *The Condition-Based Approach*

Given a distributed problem, the *condition-based* approach analyzes restrictions of the problem to subsets of its inputs. For each such restriction we obtain a new distributed problem, which is possibly easier than the original problem, since a protocol has to deal only with a subset of the possible inputs; indeed, a protocol that solves the original problem also solves the restricted problem. We are interested in computability and efficiency aspects of this approach:

—Given an unsolvable problem, for what restrictions of its inputs does the problem become solvable?

—Given a solvable problem, for what restrictions of its inputs the problem becomes easier to solve?

In this article, we consider only the first question, for restrictions of the consensus problem. In Mostefaoui et al. [2001b], we investigate the second question, and in Mostefaoui et al. [2002], we consider restrictions of the set agreement problem.

As explained in the Introduction, the motivation for the condition-based approach is to study problems in environments where not all inputs have the same probability of occurring. If we can identify a set of inputs that arrive much more frequently than others, it might be possible to design a protocol that solves the problem more efficiently. Now, to be more useful in practice, it is desirable to design a protocol that solves the restricted problem in a *safe* way. Namely, we want the protocol to be able to deal with the other inputs, which, although improbable, can nevertheless occur from time to time. In those rare situations we do not want the protocol to output arbitrary values. Also, if there are no faults, we would like the protocol to terminate even in those rare situations.

We proceed in the next section to define formally the condition-based approach for consensus, and its safe version. In the following section, we define it for general problems, and prove that a safe version of a subproblem is solvable if and only if the nonsafe version is solvable.

[1] This property is sometimes named "Uniform Agreement," in contrast to the "agreement" property requiring only that no two correct processes decide differently. These two properties are equivalent in the asynchronous systems we consider [Guerraoui 1995].

3.1. CONSENSUS AND THE CONDITION-BASED APPROACH. The input values of a protocol define a vector with one component per process, representing its private (unknown to the other processes) input to the computation. In particular, for the consensus problem, the proposed values in an execution are represented as an *input vector*, such that the ith entry contains the value proposed by p_i, or $\perp$ if p_i did not take any steps 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 $\mathcal{V}$, and $\mathcal{V}_f^n$ be the set of all the vectors with at most f entries equal to $\perp$. For $I \in \mathcal{V}^n$, let $\mathcal{I}_f$ be the set of possible views, that is, 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.

In the *condition-based* approach, we consider subsets C of $\mathcal{V}^n$, called *conditions*, that represent the common input vectors of 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. In the classic consensus problem, the trivial condition $C = \mathcal{V}^n$ is assumed, and this problem is unsolvable for every $f \geq 1$ [Fischer et al. 1985], and trivially solvable for $f = 0$. Other conditions make the problem solvable for every value of f, such as $C1$ and $C2$ mentioned in the introduction (studied in more detail in Section 6).

We say that an *f-fault tolerant protocol solves the consensus problem for a condition C* if in every execution whose input vector J belongs to $\mathcal{V}_f^n$, the protocol satisfies the following properties:

—Validity. A decided value is a proposed value.

—Agreement. No two processes decide different values.

—Guaranteed_Termination. If (1) $J \in \mathcal{C}_f$ and no more than f processes crash, or (2.a) all processes are correct, or (2.b) a process decides, then every correct process decides.

The first two are the validity[2] and agreement requirements of the classic consensus problem, and are independent of a particular condition C. The third requirement, requires termination under "normal" operating scenarios, including inputs belonging to C, and failure-free executions. Part (1), requires termination even in executions where some processes crash initially and their inputs are unknown to the other processes. This is represented by a view J with $\perp$ entries for those processes. Termination is required if it is possible that the full input vector belongs to C, that is, if J can be extended to an input vector $I \in C$. Part (2) defines two well-behaved scenarios where a protocol should terminate even if the input vector does not belong to C.[3]

[2] That is, if $J \in \mathcal{V}_f^n$ then the decided value is equal to one of the entries in J. In Attiya and Avidor [2002], a process is allowed to decide a value a not in J provided every extension of J in C contains a.

[3] One could envision other termination requirements for inputs not in C. In fact, Theorem 4.8 describes other scenarios where our protocol terminates. More about this in the Conclusion section.

3.2. SAFE VS. NONSAFE VERSIONS OF A PROBLEM. In this article, we are interested in characterizing the conditions C for which there exists an f-fault tolerant protocol that solves the consensus problem. This is a *safe* version of the problem because the protocol is required to deal also with input vectors not in C. Consider the *nonsafe* version of the problem where it is assumed that the protocol never gets inputs outside of C. This means that the protocol satisfies agreement, validity and termination in every execution of the protocol whose input vector J belongs to $\mathcal{C}_f$ (and these are the only input vectors allowed). We show here that consensus is solvable for C in the safe version of the problem if and only if it is solvable for the nonsafe version of the problem. It is not hard to generalize this result for set-consensus [Chaudhuri 1993] and other convergence tasks [Borowsky et al. 2001; Herlihy and Rajsbaum 1997], but for the purposes of this article it is sufficient to present the result for consensus only.

This equivalence of safe and nonsafe versions of the problem[4] implies that the additional safety requirements to condition based consensus (to deal with inputs not in C) do not change the characterization of solvable conditions. Thus, the condition-based approach for consensus (and for other agreement tasks, since the proof is similar) can in principle be studied with techniques that have been developed for decision tasks. For instance, to find out if consensus is solvable for a given condition, in the wait-free case, we could use the characterization theorem of Herlihy and Shavit [1999] that tells which decision tasks are wait-free solvable. However, this is in general undecidable [Gafni and Koutsoupias 1999; Herlihy and Rajsbaum 1997]. As we show in this article, for condition-based consensus problems, there is a simple decidable solvability characterization, for any value of f.

We proceed to show how to transform a protocol that solves the nonsafe version of consensus for C, into a protocol that solves it safely. We use a similar technique later on in our general condition-based consensus protocol.

THEOREM 3.1. *Consensus is solvable for C if and only if it is solvable for the nonsafe version of the problem.*

PROOF. If there is an f-fault tolerant protocol solving consensus for a condition C, then the same protocol nonsafely solves consensus for C, since the requirements for nonsafe consensus are weaker. Next we prove that if there is a protocol P solving nonsafely consensus for C, there is also a safe solution.

The argument consists of modifying the nonsafe protocol P as follows (for the purpose of the proof, we do not make any efforts of making it efficient). The idea is to execute P while the input vector belongs to $\mathcal{C}$. If at some point it is discovered that it does not belong to $\mathcal{C}$, P should not be executed anymore, since P was not designed to deal with inputs outside of $\mathcal{C}$. If a process discovers that the input does not belong to $\mathcal{C}$, it announces to everybody that it is unable to decide using P, and the control jumps to a subroutine that waits until (i) somebody decides (and if so decides that value), or until (ii) every process announces that it is unable to decide using P. In case (ii), the full input vector I is known and the process decides $F(I)$ using some fixed function F that returns one of the values in I, such as say max.

[4] This equivalence does not hold for other forms of non-safe tasks, such as the one in Attiya and Avidor [2002], where a process p_i is allowed to decide a value a not in the input vector J whenever p_i knows that a must be proposed (if nobody fails), that is, if every extension I of J in C contains a.

```
Function Safety_Routine
(1) repeat forall j ∈ [1..n] do W_i[j] ← read(O[j]) enddo;
(2)        if (∃ j : W_i[j] ≠ ⊥, ⊤) then return(W_i[j]) endif
(3) until (⊥ ∉ W_i)   endrepeat;
(4) forall j ∈ [1..n] do Y_i[j] ← read(I[j]) enddo;
(5) return(F(Y_i))
```

FIG. 1. Safety routine (for a process when it discovers that the input does not belong to C).

In more detail, in the modified protocol, P', a process p_i first writes its input value to a shared variable, $I[i]$ (assumed not to be used by P, and initialized with $\perp$), before executing P. Then p_i executes P, and when p_i decides on a value, it writes it to a shared variable $O[i]$ (assumed not to be used by P, and initialized with $\perp$). While p_i is executing P, each time it executes an instruction of P, it first snapshots the shared array $I[1..n]$. If it contains a view J in $\mathcal{C}_f$ (or J contains more than f entries equal to $\perp$), it continues executing P. Otherwise, it stops executing P, does not decide according to P, and writes $\top$ to $O[i]$, and jumps to the code in Figure 1.

When the process is about to take a decision according to P, say d, it first snapshots the shared array $I[1..n]$ of the input values written so far. If the outcome is a view J, and $J \in \mathcal{C}_f$, then the process decides d and writes the value to $O[i]$.

To prove the correctness of P', first notice that validity follows easily from the fact that P satisfies validity, and that F returns one of the values in its argument.

Agreement property of P' for input vectors in $\mathcal{C}_f$ follows from agreement of P, and the fact that in this case, the processes run P and do not jump to the safety routine. If the input J is not in $\mathcal{C}_f$, it may still be the case that a process sees a view of J that does belong to $\mathcal{C}_f$. Any two such process will have views of the input vector that are ordered by containment, and hence corresponding to an execution of P where the input vector belongs to $\mathcal{C}_f$. Thus, they decide on the same value, by the agreement property of P. Moreover, a process that sees a view of J (possibly J itself) that does not belong to $\mathcal{C}_f$, will jump to the safety routine, which guarantees that the decision value will be equal to a decision taken by a process in the first case. Or if there is no such process, and every process sees a view of J that does not belong to $\mathcal{C}_f$, they will all decide according to F, the same value.

Guaranteed_Termination is proved in a similar way. Processes that see an input vector that belongs to $\mathcal{C}_f$ terminate because P satisfies Termination. Other processes terminate if either some process decides in P, or no process was able to decide in P and nobody fails (so everybody writes its input value to I). □

4. *Acceptability and a Generic Protocol*

We start by identifying a class of conditions, called acceptable conditions, and then describe a protocol that solves consensus for any condition C in this class. Given a condition C for n processes, a fault-tolerance parameter f, and a set of input values $\mathcal{V}$, P is a predicate defined on $\mathcal{V}_f^n$, and S is a function defined on (not necessarily all of) $\mathcal{V}_f^n$. A condition C is acceptable if there exist P and S satisfying the three conditions defined below. Our condition-based consensus protocol works for any acceptable condition C, when instantiated with corresponding P and S.

The following notation will be useful in the rest of the article. Consider the partial order on vectors of $\mathcal{V}_f^n$ defined as follows $J1 \leq J2$ if $\forall k: J1[k] \neq \perp \Rightarrow J1[k] = J2[k]$. Then $|J|$ denotes the number of entries in J different from $\perp$.

4.1. ACCEPTABILITY. Here we define the three acceptability properties such that, if a condition C has a predicate P and a function S that satisfy them, then C is acceptable. Intuitively, our protocol uses P and S to solve consensus for an acceptable C, and the acceptability properties help it to enforce the Guaranteed_Termination, Agreement, and Validity requirements of condition-based consensus. In the first acceptability property, a process uses its current view J of the input vector to test (by evaluating $P(J)$) if the input vector could belong to C. Thus, P returns true at least for each vector J with at most f unknown entries, and such that J can be extended to a vector I, $I \in C$. The second property states that if two processes decide based on their views $J1$, $J2$, using S, and $J2$ contains at least as many inputs as $J1$ (and agree in the non-$\perp$ coordinates), then the decision will be the same, provided $P(J1)$, $P(J2)$ hold. The third property enforces the validity requirement of consensus. Given a condition C, the properties that its P and S have to satisfy are formally defined as follows:

—Property $\mathrm{T}_{C\to P}$: $I \in C \Rightarrow \forall J \in \mathcal{I}_f : P(J)$.
—Property $\mathrm{A}_{P\to S}$: $\forall J1, J2 \in \mathcal{V}_f^n : (J1 \leq J2) \wedge P(J1) \wedge P(J2) \Rightarrow S(J1) = S(J2)$.
—Property $\mathrm{V}_{P\to S}$: $\forall J \in \mathcal{V}_f^n : P(J) \Rightarrow S(J)$= a non-$\perp$ value of J.

Definition 4.1. A condition C is *f-acceptable* if there exist a predicate P and a function S satisfying the properties $T_{C\to P}$, $A_{P\to S}$ and $V_{P\to S}$ for f. Any such P, S are said to be *associated* with C for f.

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

Notice that acceptability (although motivated by the three consensus requirements) is a purely combinatorial property of the set C. For example, it is not hard to check that there are no P, S satisfying the three acceptability properties for $C = \mathcal{V}^n$ when $f \neq 0$. Indeed, if such a C was an acceptable condition, then the protocol proposed next would solve the consensus problem despite process crashes, contradicting the impossibility result of Fischer et al. [1985].

Our first main result is the following theorem. It is proved in the next section, by presenting a generic protocol and proving it correct.

THEOREM 4.2. *If C is f-acceptable, then there exists an f-fault tolerant protocol solving consensus for C.*

4.2. THE PROTOCOL. The protocol in Figure 2 solves consensus for any f-acceptable condition C with associated parameters P and S. It has to be instantiated with P and S. In addition, the protocol has to be instantiated with a function F from $\mathcal{V}^n$ to $\mathcal{V}$ which returns a fixed, arbitrary value of the input vector I. Any such function will do. Thus, F does not depend on the particular values of P and S. The protocol uses shared registers and local variables. Those are first introduced. Then, the protocol behavior is described and proved correct.

Shared memory. The shared memory is made up of two arrays of atomic registers $V[1\cdots n]$ and $W[1\cdots n]$, both initialized to $[\perp, \ldots, \perp]$. Their meaning

```
Function SM_Consensus(v_i)
(1)  write(v_i, V[i]);
(2)  repeat V_i ← snapshot(V) until (#(V_i[j] ≠ ⊥) ≥ (n − f)) endrepeat;
(3)  if P(V_i) then w_i ← S(V_i) else w_i ← ⊤ endif;
(4)  write(w_i, W[i]);
(5)  repeat forall j ∈ [1 · · n] do W_i[j] ← read(W[j]) enddo;
(6)         if (∃ j : W_i[j] ≠ ⊥, ⊤) then return(W_i[j]) endif
(7)  until (⊥ ∉ W_i) endrepeat;
(8)  forall j ∈ [1 · · n] do Y_i[j] ← read(V[j]) enddo;
(9)  return(F(Y_i))
```

FIG. 2. A generic condition-based consensus protocol.

is the following:

—$V[i]$ is the shared register where p_i deposits its input value v_i (line 1).

—$W[j]$ is the shared register where p_i deposits (in line 4) p_j's estimate of the decision value (w_j). If the local view of p_j does not allow for the possibility that the input vector is in C then p_j's estimate is set to a default value $\top \notin \mathcal{V} \cup \{\bot\}$ (line 3).

Local variables. Each process p_i manages three arrays of local variables where it stores a local copy of its current view of the state of the shared memory.

—$V_i[1 \cdots n]$ is an array where p_i builds its local view.
A process p_i can find out the current state of proposals by invoking snapshot(V). At line 2, $\#(V_i[j] \neq \bot)$ denotes the number of non-$\bot$ entries of V_i. We say that a process gets its "local view," when it gets an array where at least $(n - f)$ values are different from $\bot$ (so the local view of a process is unique, if any).

—$W_i[1 \cdots n]$ is an array contains the decision value estimates of each process p_j.

—$Y_i[1 \cdots n]$ is an array used by p_i to store the values proposed by each process.

The protocol has a three-part structure.

Part 1 (*lines* 1–2). A process p_i first writes its input value v_i to its entry of the shared array V. Then p_i repeatedly snapshots V until at least $(n - f)$ processes (including itself) have written their input values in V; its view V_i contains the result of the last snapshot, where $V_i[j]$ is the input value of p_j, or $\bot$ if p_j has not yet written its input value.

Part 2 (*lines* 3–4). Now, p_i enters its wait-free, condition-dependent, part of the protocol. With its view V_i, it tries to make a decision, by evaluating $P(V_i)$. If true, p_i will be able to decide $S(V_i) = w_i$ after having written the value it decides w_i (or $\top$ if it could not decide) in the shared array $W[i]$ to help other processes decide in the next part.

Part 3 (*lines* 5–9). Finally, p_i enters a loop to look for a decision value (i.e., a value different from $\bot, \top$) provided by a process p_j (possibly itself) in the shared array $W[j]$. 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 (8)) to get the full input vector, and proceed to decide according to the fixed, deterministic rule F.

4.3. CORRECTNESS. We proceed to prove Theorem 4.2 by showing that the generic protocol solves the consensus problem for any acceptable condition C, assuming it has been instantiated with associated P, S. That is, we show that it satisfies Validity, Agreement and Guaranteed_Termination.

Termination

LEMMA 4.3. *The protocol satisfies part* (1) *of* Guaranteed_Termination (*namely, if the input vector* $J \in \mathcal{C}_f$ *and there are at most* f *crashes, then each correct process decides in line* 6).

PROOF. 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 local view V_i, $V_i \leq J$. Since $J \in \mathcal{C}_f$, also $V_i \in \mathcal{C}_f$, and it follows from $\mathrm{T}_{C \to P}$ that $P(V_i)$ is true, and from line (3) that $w_i \neq \perp, \top$. Hence, $W[i] \neq \perp, \top$. Consequently, at line (6), $W_i[i] \neq \perp, \top$ holds, and this allows p_i to decide in this line. □

LEMMA 4.4. *The protocol satisfies part* (2) *of* Guaranteed_Termination *(namely, if* (2)(a) *all processes are correct, or* (2)(b) *a process decides, then every correct process decides).*

PROOF. Let us first assume that all processes are correct. They all execute line (1), and hence they all exit the loop of line (2). If all processes evaluate to false P in line (3), then they all eventually read $\top$ everywhere in line (5), and they all eventually decide in line (9).

Let us assume that a process p_i decides. Then p_i exits the loop of line (2), and hence at least $n - f$ processes execute line (1). Thus, every correct process exits the loop of line (2). Now, if p_i decides in line (6), then some process p_j finished executing line (4) with a value different from $\perp, \top$. Thus, every correct process will eventually see this value in line (5), and decide in line (6) (if not before). Otherwise, p_i decides in line (9), and hence it sees $\top$ everywhere in its local array W_i, which implies that every process has executed line (4) writing $\top$. And every correct process will eventually see all these values, and terminate also in line (9). □

Agreement

LEMMA 4.5. *Either all processes that decide do it in line* (6) *or in line* (9).

PROOF. We consider two cases. A process p_i decides in line (6) in the first case, and in line (9) in the second case.

—In the first case, process p_i sees a value different from $\perp, \top$ in line (6), and hence for some j, $W[j] \neq \perp, \top$. As $W[j]$ is initialized to $\perp$, and $W[j]$ is written only once, no processor will ever see a value $\top$ for position j in line (6). Now, a process exists the loop in line (7) only when it sees $\top$ in all the positions of its variable W, and therefore no process will exit the loop in this line. It follows that if a process decides, it will do it in line (6).

—In the second case, a process does exit the loop in line (7), and hence every process has evaluated to false P in line (3) and written $\top$ in the shared array W in the next line. Thus, no process will decide in line (6). □

LEMMA 4.6. *The protocol satisfies* Agreement *(no two processes decide different values).*

PROOF. Let us consider two processes p_i and p_j that decide. By Lemma 4.5, they decide in the same line.

—Let us assume that both processes decide at line (6): p_i decides $W_i[\ell] = w_\ell$, while p_j decides $W_j[k] = w_k$. It follows that there exist two local views V_ℓ and V_k such that p_ℓ has computed $S(V_\ell) = w_\ell \neq \perp, \top$, while p_k has computed $S(V_k) = w_k \neq \perp, \top$. This means that both $P(V_\ell)$ and $P(V_k)$ are satisfied (I1).

The last invocations of snapshot(V) in line (2) by p_ℓ and p_k have defined their local views V_ℓ and V_k, respectively. Moreover, since snapshots can always be ordered by containment, we conclude $V_\ell \leq V_k$ or $V_k \leq V_\ell$ (I2). It follows from (I1), (I2) and the property $\mathrm{A}_{P \to S}$ that $S(V_\ell) = S(V_k)$, that is, $w_\ell = w_k$.

—Assume that both p_i and p_j decide at line (9). In that case, each process p_ℓ has executed line (4) and consequently $W[\ell] \neq \perp$. As p_ℓ executes line (1) before line (4), it follows that $V[\ell] = v_\ell$ when p_i (or p_j) executes line (8). Hence, $Y_i = Y_j = (v_1, \ldots, v_n)$. Since both processes apply the same deterministic function F to the same vector, they get the same result value. □

Validity

LEMMA 4.7. *The protocol satisfies* Validity *(a decided value is a proposed value).*

PROOF. There are two cases according to the line at which a process decides.

—Let us first consider the case of a process p_i that decides at line (6) by returning the value $W_i[j]$. As we have $W_i[j] = w_j \neq \perp, \top$, we conclude that V_j, the local view of p_j, is such that $P(V_j)$ is true and $w_j = S(V_j)$. The validity follows from the property $\mathrm{V}_{P \to S}$, associated with P and S.

—Let us now consider the case of a process p_i that decides at line (9). Then, we have $\forall j : W_i[j] \neq \perp$ (line (7)) from which we conclude that each process p_j has deposited its value v_j into $V[j]$. Hence, we have $Y_i = [v_1, \ldots, v_n]$ at line 8. As F outputs a value of Y_i, the validity property is satisfied. □

This completes the proof of Theorem 4.2.

4.4. MORE ON TERMINATION. We have shown in Lemmas 4.3 and 4.4 that, for any given f-acceptable condition C with associated P, S, the generic protocol of Figure 2 terminates in all cases described by the Guaranteed_Termination property. Actually, there are other situations where correct processes terminate, even if the input vector does not belong to C or the number of failures exceeds f.

First, it is easy to conclude from the protocol text that if all the processes deposit a value in the shared array W (i.e., each process executes line (4)), then the correct process decide, no matter what the number of failures is (processes that crash will do so after line (4)), and even if the input vector does not belong to C.

Second, consider an input vector I that does not belong to C, but that it contains a vector J, $J \in \mathcal{I}_f$, with an extension I' in C: $J \leq I' \in C$. There are executions where the protocol will terminate, even if some processes know that the input vector I is not in C. Correct processes will terminate in all executions where at least one

process does not know that I is not in C. More precisely, correct processes will terminate in all executions where the local view V_j of a process p_j in line (2) is equal to J, and p_j executes line (4) (so at this point p_j considers possible that the input vector is I' instead of I). This follows from the code and because Property $T_{C \to P}$ implies that $P(J)$ is true.

After the previous discussion it is not hard to prove the next theorem.

THEOREM 4.8. P-More_Termination. *All correct processes decide if and only if (a) all the processes execute line* (4)*, or (b) the local view of a process p_i that executes line* (4) *has an extension in C.*

The termination conditions of this theorem are about a particular protocol, while the Guaranteed_Termination conditions are about the consensus problem. Notice that these two conditions are related as follows:

—Guaranteed_Termination (2a) or (1) implies (a) or (b), and

—Guaranteed_Termination (2b) implies (a) or (b)"

For an example of the second termination situation in the theorem, consider the following situation: $\mathcal{V} = \{0, 1, 2\}$, $n = 6$, $f = 2$, $I = [1, 1, 0, 0, 2, 2]$, and the condition is $C2$ (this condition, presented in the Introduction and studied in Section 6.3, favors the maximal value present in the vector). More precisely, $I \in C2$ iff $[a = \max(I) \Rightarrow \#_a(I) > f]$ (where $\#_a(I)$ denotes the number of occurrences of a in the vector I). Thus, $I \notin C2$. Assume that a process p_j gets the local view $V_j = [1, 1, 0, 0, \perp, \perp]$. Since $I' = [1, 1, 0, 0, 1, 0]$ is an extension of V_j that belongs to $C2$, p_j evaluates $P(V_j) = \mathit{true}$ and decides the value $S(V_j) = 1$ accordingly. Consequently, all the correct processes also decide.

These additional termination situations are interesting from both practical and theoretical points of view, showing that the correct processes "almost always" terminate in this protocol. Interestingly, the more entries of a local view V_j that are equal to $\perp$, the more possibilities for V_j to have an extension that belongs to C and hence for processes to terminate. Less information—slow processes—can help processes decide! For example, notice that in condition $C2$ and $C1'$ described below, a process that gets a local view with f entries equal to $\perp$ *always* decides.

5. *A Characterization of the Conditions for Consensus Solvability*

In this section we prove our second main result, the converse to Theorem 4.2:

THEOREM 5.1. *If there exists an f-fault tolerant protocol solving consensus for C, then C is f-acceptable.*

Proving this theorem completes a characterization of the conditions allowing a consensus solution. We prove this theorem using an alternative form of the acceptability definition, called legality. The final result is stated in Theorem 5.7 (which implies Theorem 5.1).

5.1. THE NOTION OF LEGALITY. Given a condition C and a value for f, consider the graph $Gin(C, f)$ (close to the graph defined in Biran et al. [1990a] for $f = 1$): Its vertices are $\mathcal{C}_f$, that is, the input vectors I of C plus all their views, $J \in \mathcal{I}_f$. Two views $J1, J2 \in \mathcal{C}_f$ are connected by an edge iff $J1 \leq J2$. Hence, two vertices $I1$, $I2$ of C are connected (by a path) if their Hamming distance $d(I1, I2) \leq f$.

The graph $Gin(C, f)$ is made up of one or more connected components, namely, $G_1, \ldots, G_x$.

Definition 5.2. A condition C is *f-legal* if, for each connected component of $Gin(C, f)$, all the vertices that belong to this component have at least one input value in common.

LEMMA 5.3. *Let C be a finite condition. It is decidable in polynomial time if C is f-legal.*

PROOF. Let us consider the graph $H(C, f)$ whose vertices are the vectors of C and there is an edge connecting $I1$ and $I2$ if $d(I1, I2) \leq f$. The vertices of $H(C, f)$ are the subset of vertices of $Gin(C, f)$ that do not include any entries equal to $\perp$. Two vertices are connected in $H(C, f)$ if and only if the corresponding vertices are connected in $Gin(C, f)$. Also, a connected component of $Gin(C, f)$ has one input value in common to all its vertices if and only if the corresponding component of $H(C, f)$ has at least one input value in common, and this value appears $f + 1$ times in each one of its vertices. Hence, given a condition C, showing that C is f-legal (in accordance with Definition 5.2) amounts to show that every connected component of $H(C, f)$ has at least one input value in common, and that this value appears $f + 1$ times in each one of its vertices.

The graph $H(C, f)$ can be constructed in time $O(n \cdot |C|^2)$: the edges are defined by comparing the n entries of each pair of vectors of C; the graph has $|C|$ vertices and $O(|C|^2)$ edges. Once the graph is constructed, the connected components can be identified in time $O(|C|^2)$, using say, BFS. Finally, the intersection of the values appearing in the vectors of a connected component H_i can be computed in polynomial time. A straightforward way of doing this is by sorting the values X of one of its vertices, and then, for every other vertex of the component, binary searching X for each of the values of the vertex. If a value does not appear in X, it is removed from X. This procedure takes time $O(n \log n \cdot |H_i|)$, where $|H_i|$ is the number of vertices in H_i. Finally, one checks that at least one of these values appears $f + 1$ times in every vector of the component. □

Assuming a legal condition, for every G_i of $Gin(C, f)$, there is a nonempty set, $d(G_i)$, of input values that appear in each input vector of the connected component G_i. As we shall see, when $I \in G_i$, the protocol actually forces the processes to deterministically decide the same value, one of $d(G_i)$. Moreover, if $a \in d(G_i)$, then $\#_a(I) \geq f + 1$ for every $I \in G_i$ with $\#_\perp(I) = 0$. Otherwise, we can replace all occurrences of a in I by $\perp$ and obtain J with $\#_\perp(J) \leq f$, and $\#_a(J) = 0$, which is impossible, since $J \in G_i$ and $a \in d(G_i)$. Thus, it is no coincidence that this property holds for $C1$ and $C2$.

5.2. FROM A CONDITION-BASED PROTOCOL TO A LEGAL CONDITION. In a *decision task* each process starts with an input value, and has to eventually decide on an output value. A decision task specifies the input vectors that can be an initial configuration for the processes, and for each one, a set of output vectors, that correspond to correct final configurations. Given a protocol that solves an arbitrary task f-resiliently, we can consider the output vector of n values decided by the processes in an execution where at most f processes crash. If the task is a consensus task, then all the non-$\perp$ entries of the vector are equal, by the agreement requirement. Let X be a set of input vectors for the task. We are interested in the graph $Gout(X, f)$, whose

vertices are all decision vectors over all executions starting on inputs X, and where two vectors $J1$, $J2$ are connected by an edge if $J1 \leq J2$. We use the following theorem, a simple extension of Theorem 3.5 in Moran and Wolfstahl [1987] (used later in Biran et al. [1990a] to characterize the solvable problems with $f = 1$). The proof in this paper is for the message passing model and $f = 1$, but the extension to our shared memory model with $f > 1$ is easy. We present this extension here for completeness, almost verbatim from Moran and Wolfstahl [1987]. We could have proved the theorem using other techniques; for example, see Moses and Rajsbaum [2002] for a proof technique that does not use a reduction to the FLP impossiblity result [Fischer et al. 1985].

THEOREM 5.4. *If $Gout(X, f)$ is the graph of decision vectors of an f-fault tolerant protocol on a connected input graph $Gin(X, f)$, then $Gout(X, f)$ is connected.*

PROOF. The proof is by reduction to the following well-known form of the FLP impossiblity result [Fischer et al. 1985]. This form of the impossibility result is for a variant of the consensus problem where the possible decision values of the processes are 0 and 1. The set of input vectors X is arbitrary, as long as it defines a connected graph $Gin(X, f)$. The validity requirement is that there are two input vectors that lead to different decision values. Agreement and termination requirements are as usual.

The proof that this form of the consensus problem is not solvable for $f > 0$ follows the same arguments of the proof in Fischer et al. [1985]: Let I, I' be two input vectors that lead to different decision values. Since the input graph is connected, there is a path from I to I'. On this path, there must be two adjacent input vectors that might lead to different decision values. From this point, the proof is identical to the one of Fischer et al. [1985] is for $f = 1$ but the generalization to $f > 0$ is obvious).

To prove the theorem, assume for contradiction there is a protocol P whose graph of decision vectors $Gout(X, f)$ is not connected. Let $G_1, G_2, \ldots, G_p$ be the connected components of $Gout(X, f)$, $p \geq 2$. We construct a protocol P^c based on P that solves the above consensus problem in spite of f faults on the same input graph $Gin(X, f)$.

The protocol P^c works as P except that when a processor is about to return a decision value according to P, it first writes the decision to the shared memory. Call this value an *intermidiate* decision value. Then, the process enters a loop where it reads the shared-memory until it sees at least $n - f$ intermidiate decision values. Since P solves the task in spite of f faults, eventually $n - f$ processes will write to the shared memory their intermidiate values. Thus, each correct process p_i will eventually construct a view J_i of intermidiate decision values, with at least $n - f$ non-$\perp$ entries. Thus, J_i is a vertex of $Gout(X, f)$. Moreover, any two such views, J_i, J_j are in the same connected component G_m of $Gout(X, f)$, since both are views of the same output vector I, $J_i \leq I$, $J_j \leq I$, where I is a vertex of G_m that corresponds to the decision vector in the corresponding execution of P. The process will return as decision for P^c the parity bit of m and halt. Since all processes agree on the same connected component G_m, they will all decide on the same parity bit. Therefore, P^c solves the above variant of consensus, a contradiction. □

For the case of a nonsafe consensus problem for a condition $\mathcal{C}$ (Section 3.2), the set of input vectors corresponds to $\mathcal{C}_f$, while the set of output vectors contains one output vector for each possible decision value $v \in \mathcal{V}$, with all its entries equal to v, and joined by an edge to all its views.

LEMMA 5.5. *If the nonsafe consensus problem for a condition C is f-fault tolerant solvable, then C is f-legal.*

PROOF. Assume there is an f-fault tolerant protocol solving nonsafe consensus for C. For each connected component G_i of the input graph, consider the graph $Gout_i$ of decision vectors of the protocol, on all executions starting with inputs of this component with at most f failures. By the termination requirement of consensus, all correct processes decide in such executions, and hence the corresponding decision vectors have at most f entries equal to $\perp$. Notice that every vertex in $Gout_i$ corresponds to a decision vector of the protocol, which must contain all non-$\perp$ entries equal to the same value, by the agreement requirement of consensus. It follows from Theorem 5.4 that the graph $Gout_i$ is connected.

We now show that all vertices of $Gout_i$ contain just one input value, say d. Consider two vertices I, I' of $Gout_i$, and assume for contradiction that every (non-$\perp$) entry of I is equal to a while every (non-$\perp$) entry of I' is equal to b. Since $Gout_i$ is connected, there is a path from I to I'. Thus, there are two adjecent vertices, J, J', in this path, one with (non-$\perp$) entries equal to a and the other equal to b. By definition of $Gout_i$, $J \leq J'$ or $J' \leq J$. This is a contradiction, since J has only a's and J' has only b's (in addition to $\perp$'s) and $a \neq b$.

We have shown that $Gout_i$ contains vertices with only one decision value, say d. By the validity property of decision consensus, d must be a value in every input vector of G_i. And since G_i is an arbitrary connected component of the input graph, then C is f-legal. □

5.3. FROM LEGALITY TO ACCEPTABILITY. We now show how, given an f-legal condition C, there is an efficient way of constructing its actual predicate P and function S used by the generic protocol, and hence C is f-acceptable.

Let $J \in \mathcal{V}_f^n$ (i.e., a vector with at most f entries equal to $\perp$.) We say that an input vector I is a *legal extension* of J, if $J \in \mathcal{I}_f$ and $I \in C$. Notice that several input vectors can be legal extensions of the same J.

For a given legal condition C, P and S are constructed as follows. For every $J \in \mathcal{V}_f^n$:

—$P(J)$ is true if there exists a legal extension of J.

—$S(J)$ = a deterministically chosen value of $d(G_i)$, where G_i is the connected component including all legal extensions of J.

Notice that S is well defined. First, $S(J)$ has to be defined only when $P(J)$ is true. Second, if I_1 and I_2 are two legal extensions of J, they belong to the same connected component G_i as they differ in at most f values. Third, since C is f-legal, $d(G_i)$ is not empty. Also, it is easy to see that all this can be executed in polynomial time, using ideas similar to the ones described for Lemma 5.3. Thus, it is easy to show the following result.

LEMMA 5.6. *Any f-legal condition C is f-acceptable, and associated P and S can be computed in polynomial time.*

PROOF. Let P and S be parameters as defined above. Trivially, P satisfies property $T_{C\to P}$. For $A_{P\to S}$, notice that for $J1$, $J2$, there is a legal extension I of both vectors, since $J1 \leq J2$. Thus, the value $S(J1)$ is equal to a deterministically chosen value of $d(G_i)$, where G_i is the connected component including I, and therefore, $S(J2)$ is equal to the same value. Finally, $V_{P\to S}$ follows from the fact that J is in G_i, the connected component including any of its legal extensions I, since J and I are joined by an edge. Also, it is not hard to check (as in Lemma 5.3) that P and S can be computed in polynomial time. □

5.4. MAIN THEOREM. The following summarizes our main results.

THEOREM 5.7. Main Theorem. *The following three assertions are equivalent, and decidable in polynomial time.*

—A1. *Condition C is f-acceptable.*

—A2. *Condition C is f-legal.*

—A3. *The nonsafe version of the consensus problem for C is f-fault tolerant solvable.*

—A4. *The consensus problem for C is f-fault tolerant solvable.*

PROOF. Theorem 4.2 shows that A1 ⇒ A4, and it is easy to see that A4 ⇒ A3, since the same protocol that solves a consensus problem solves its nonsafe version. Lemma 5.5 shows that A3 ⇒ A2, Lemma 5.6 shows that A2 ⇒ A1. Lemma 5.3 shows that A2 is decidable in polynomial time, and hence so are the two other assertions. □

It is interesting that consensus solvability can be decided in polynomial time for every C and f, while the general problem of deciding if a distributed problem is f-fault tolerant solvable is undecidable [Gafni and Koutsoupias 1999; Herlihy and Rajsbaum 1997] when $f > 1$. Even in the case of $f = 1$ the general decidability problem is difficult: it was shown to be NP-hard in Biran et al. [1990b].

The previous theorem is analogous to the characterization of the class of weakest failure detectors for consensus [Chandra et al. 1996], in the sense that this class identifies the minimal properties that a failure detector D must satisfy for consensus to be solvable. The previous theorem does the same for conditions: it characterizes the largest set of conditions for consensus to be solvable.

6. *Two Conditions*

This section presents the two conditions described in the introduction, $C1$ and $C2$, and proves them to be acceptable, by defining associated predicates P and functions S. Both conditions are parameterized by f. Once we have developed the required tools of Section 5, $C1'$ (a slight refinement of $C1$) and $C2$ are proved to be maximal in Section 7.

Notation. $\#_a(J)$ denotes the number of entries of J that are equal to a, where J is a vector in $\mathcal{V}_f^n$, and $a \in \mathcal{V} \cup \{\perp\}$.

6.1. CONDITION $C1$. We use the following definitions for $C1$, where $J \in \mathcal{V}_f^n$, and $\widehat{J}$ is the vector obtained from J by choosing a non-$\perp$ value a of J that appears the most often, and replacing it by $\perp$.

—$\#_{1st}(J) = \max_{a \in \mathcal{V}} \#_a(J)$.
—$\#_{2nd}(J) = \#_{1st}(\widehat{J})$.

That is, $\#_{1st}(J)$ returns the number of occurrences in J of a non-$\perp$ value that appears the most often in J. Notice that there may be more than one value that appears the most often in J, if such values appear equal number of times. In this case $\#_{1st}(J)$ returns that number (for one of those values, such as the smallest, for instance). Thus, if J is a vector with at least one non-$\perp$ entry, we have:

—J contains a single non-$\perp$ value iff $\#_{2nd}(J) = 0$ (hence $\#_{1st}(J) = n - \#_\perp(J)$).
—There are several non-$\perp$ values that appear the most often in J iff $\#_{2nd}(J) = \#_{1st}(J)$.
—J contains at least two different values and there is a single (non-$\perp$) value that appears the most often in J iff $\#_{2nd}(J)$ returns the number of occurrences in J of a non-$\perp$ value that appears the second most often in J (thus, $\#_{2nd}(J) < \#_{1st}(J)$).

With these notations $C1$ for f is stated as follows, for any vector $I \in \mathcal{V}^n$:

$$\boxed{(I \in C1) \ \textit{iff} \ \ [\#_{1st}(I) - \#_{2nd}(I) > f].}$$

The intuition is that, when this condition is satisfied, a process can decide the value it has seen most often, despite up to f process crashes.

Let $I \in \mathcal{V}^n$ and $J \in \mathcal{I}_f$. Condition $C1$ has associated parameters:

—$P1(J) \equiv \ \#_{1st}(J) - \#_{2nd}(J) > f - \#_\perp(J)$.
—$S1(J) = \ a : \#_a(J) = \#_{1st}(J)$.

The following theorem shows that $P1$ and $S1$ can be used to instantiate the consensus protocol, and hence that the problem is solvable for $C1$ and f.

THEOREM 6.1. *$C1$ is f-acceptable with associated parameters $P1$, $S1$.*

PROOF. $\mathrm{T}_{C1 \to P1}$: We have to show that $I \in C1 \Rightarrow \forall J \in \mathcal{I}_f : P1(J)$. That is,

$$\#_{1st}(I) - \#_{2nd}(I) > f \ \Rightarrow \forall \ J \in \mathcal{I}_f \ : \ \#_{1st}(J) - \#_{2nd}(J) > f - \#_\perp(J).$$

Assume I satisfies $\#_{1st}(I) - \#_{2nd}(I) > f$, and consider a $J \in \mathcal{I}_f$. Assume there are x_1 vector positions such that in J they are equal to $\perp$, while in I they are equal to the value that is most common in I. Similarly, assume there are x_2 vector positions such that in J they are equal to $\perp$, while in I they are equal to a value that is the second most common in I. Thus, $x_1 + x_2 \leq \#_\perp(J) \leq f$. Now, notice that if a is a value that is the most common in I, it will still be most common in J, since $\#_{1st}(I) - \#_{2nd}(I) > f$ and at most f entries of I are changed to $\perp$ in J. Thus, $\#_{1st}(J) \geq \#_{1st}(I) - x_1$. Also, $\#_{2nd}(J) \geq \#_{2nd}(I) - x_2$. Hence, $\#_{1st}(J) - \#_{2nd}(J) \geq \#_{1st}(I) - \#_{2nd}(I) - x_1 + x_2$. And we are assuming $\#_{1st}(I) - \#_{2nd}(I) > f$, so $\#_{1st}(J) - \#_{2nd}(J) > f - x_1 + x_2$. This gives the result since $x_2 \geq 0$ and $x_1 \leq \#_\perp(J)$, as observed above.

$\mathrm{A}_{P1 \to S1}$: Consider two vectors $J1$ and $J2$ of $\mathcal{V}_f^n$ such that $(J1 \leq J2) \wedge P1(J1) \wedge P1(J2)$. We have to show that $S1(J1) = S1(J2)$.

If a is a value that is the most common in $J2$, it will still be most common in $J1$, and the proof follows. To see this, first notice that since $J1 \leq J2$, some number x of entries of $J2$ are changed to $\perp$ to create $J1$. Clearly, $x \leq f - \#_\perp(J2)$,

since in $J2$ already $\#_{\perp}(J2)$ entries equal $\perp$. Therefore, $P(J2) \equiv \#_{1st}(J2) - \#_{2nd}(J2) > f - \#_{\perp}(J2)$ implies that a is also the most common value of $J1$.

$V_{P1 \to S1}$: This property is trivially satisfied because $S1(J) =$ the most common non-$\perp$ value of J (recall that $f < n$). □

The case of binary consensus. In the binary consensus problem, $| \mathcal{V} | = 2$. In this case, $\#_{2nd}(I) = n - \#_{1st}(I)$. Thus, $C1$ can be written as $I \in C1_{bin} \equiv \#_{1st}(I) > (n + f)/2$.[5] Also, since $\#_{1st}(J) = n - \#_{2nd}(J) - \#_{\perp}(J)$, the associated parameters $P1$, $S1$ can be written as $P1_{bin}(J) = \#_{2nd}(J) < (n - f)/2$, and $S1_{bin}(J) = \#_{1st}(J)$. Thus, in the binary case, Theorem 6.1 shows that $C1_{bin}$ is acceptable with associated parameters $P1_{bin}$ and $S1_{bin}$.

6.2. $C1'$: A REFINEMENT OF $C1$. Given a condition C, a natural question is if C can be extended to include more vectors, and still allowing a consensus solution. Alternately, we would like to be able to prove that a condition C is *maximal* in this sense. We will answer this question later on, but meanwhile we can see that $C1$ is not maximal, because there is a condition $C1'$ that is f-acceptable with associated parameters $P1'$, $S1'$ (the proof is similar and is omitted), and $C1 \subset C1'$. In Section 7, we prove that $C1'$ is maximal. This refined condition assumes a total order (denoted $<$) on the values of $\mathcal{V}$. It is formally defined as follows. For any vector $I \in \mathcal{V}^n$:

$$
\begin{aligned}
(I \in C1') \quad \textit{iff} \quad & [(I \in C1) \vee \\
& ((\#_{1st}(I) - \#_{2nd}(I) = f = 0) \vee \\
& ((\#_{1st}(I) - \#_{2nd}(I) = f > 0) \wedge \\
& [\forall a, b : (\#_a(I) = \#_{1st}(I)) \wedge (\#_b(I) = \#_{2nd}(I)) \Rightarrow a < b]) \\
&) \\
&].
\end{aligned}
$$

The intuition is that $C1$ can be refined in the case where ties are encountered, by preferring smaller values. In the case I is such that $\#_{1st}(I) - \#_{2nd}(I) = f$, it is possible that a process does not see f entries of I with value a such that $\#_a(I) = \#_{1st}(I)$. In this situation, the process has a view J where $\#_{1st}(J) = \#_{2nd}(J)$, and it does not know if the missing entries are equal to a. Hence, it does not know if the original input vector was I, or the missing entries are equal to b, where $\#_b(J) = \#_{2nd}(J)$, and the input vector was another vector I'. The solution in $C1'$ is to include only one of these input vectors, I and not I', with an additional constraint on a and b: $a < b$.

Arguments similar to the ones for $C1$ can be used to show that the parameters associated with $C1'$ are:

—$P1'(J) \equiv P1(J) \vee$
$[(\#_{1st}(J) - \#_{2nd}(J) = f - \#_{\perp}(J) = 0) \vee$
$((\#_{1st}(J) - \#_{2nd}(J) = f - \#_{\perp}(J) > 0) \wedge [\forall a, b : (\#_a(J) = \#_{1st}(J) \wedge \#_b(J) = \#_{2nd}(J)) \Rightarrow a < b])]$.

—$S1'(J) = \min\ a\ :\ \#_a(J) = \#_{1st}(J)$.

Notice that when $f = 0$ every vector is in $C1'$ and $P1'$ is always true.

[5] Taubenfeld and Moran [1996] showed that binary consensus is solvable using a condition equivalent to this one.

6.3. THE CONDITION $C2$. The idea for $C2$ is to guarantee that all processes have the same extremal (e.g., largest or smallest) value in their local views. We (arbitrarily) consider the largest value, and include in $C2$ every vector whose largest input value appears more than f times. The formal definition is the following, where $\max(I)$ denotes the largest (non-$\perp$) value contained in the input vector I:

$$\boxed{(I \in C2) \ \textit{iff} \ [\#_{\max(I)}(I) > f].}$$

To define $P2$ and $S2$ we fix a value of f, and consider vectors $J \in \mathcal{V}_f^n$.

—$P2(J) \equiv \#_{\max(J)}(J) > f - \#_{\perp}(J)$.
—$S2(J) = \max(J)$.

THEOREM 6.2. *$C2$ is f-acceptable with associated parameters $P2$ and $S2$.*

PROOF. $T_{C2 \to P2}$: Assume $I \in C2$. Hence $\#_a(I) > f$ for the largest non-$\perp$ value $a \in I$. Consider any $J \in \mathcal{I}_f$. Notice that x values are changed to $\perp$ from I to J, $x = \#_{\perp}(J)$. Thus, in particular, $\#_a(J) \geq \#_a(I) - x = \#_a(I) - \#_{\perp}(J)$. Therefore, $\#_a(J) > f - \#_{\perp}(J)$, since $\#_a(I) > f$, and $P2(J)$ holds.

$A_{P2 \to S2}$. Consider $J1, J2 \in \mathcal{V}_f^n$. We show that if $J1 \leq J2 \wedge P2(J1) \wedge P2(J2)$, then $\max(J1) = \max(J2)$, i.e., the largest non-$\perp$ values in $J1$ and $J2$ are the same. Since $S2$ chooses this value, $S2(J1) = S2(J2)$. Assume $a = \max(J2)$. The claim will follow if we prove that $\#_a(J1) > 0$, because $J1 \leq J2$.

Since $P2(J2)$, we have $\#_a(J2) > f - \#_{\perp}(J2)$. Notice that x values are changed to $\perp$ from $J2$ to $J1$, $x = \#_{\perp}(J1) - \#_{\perp}(J2)$. Thus, $\#_a(J1) \geq \#_a(J2) - x = \#_a(J2) - \#_{\perp}(J1) + \#_{\perp}(J2)$. Since $\#_a(J2) > f - \#_{\perp}(J2)$, we have $\#_a(J1) > f - \#_{\perp}(J2) - \#_{\perp}(J1) + \#_{\perp}(J2) = f - \#_{\perp}(J1)$. And the claim follows, because $\#_{\perp}(J1) \leq f$ implies $\#_a(J1) > f - \#_{\perp}(J1) \geq 0$.

$V_{P2 \to S2}$. Follows directly from the fact that $\forall J \in \mathcal{V}_f^n$, J includes a non-$\perp$ value (recall that $f < n$). □

6.4. NONCOMPOSABILITY OF CONDITIONS. The following theorem shows that the set of f-acceptable conditions is not union-closed: the union of f-acceptable conditions does not always define an f-acceptable condition. This is not surprising. We show in Section 7 that $C1'$ and $C2$ are maximal: thus there union cannot be acceptable. Nevertheless, we provide here a different proof, that in addition shows that $C1$ and $C2$ may be complementary in the sense of the following lemma.

LEMMA 6.3. *Let $\mathcal{V} = \{a, b\}$, and consider any n, f such that $f < n/3$. Then for all $I \in \mathcal{V}^n$ we have that $I \in C1$ or $I \in C2$.*

PROOF. Let $I \in \mathcal{V}^n$ such that $I \notin C1$. We show that $I \in C2$. $I \notin C1 \Rightarrow \#_a(I) - \#_b(I) \leq f \wedge \#_b(I) - \#_a(I) \leq f$. This means that $-f \leq \#_a(I) - \#_b(I) \leq f$. Let

$$\#_a(I) = \#_b(I) + x \tag{1}$$

with $-f \leq x \leq f$ and without loss of generality let $a > b$. We have $\#_a(I) + \#_b(I) = n > 3f$ from which we get $\#_a(I) > 3f - \#_b(I)$. This equation combined with (1) gives: $\#_a(I) > 3f - \#_a(I) + x$ or equivalently $2\#_a(I) > 3f + x$. Since $-f \leq x \leq f$, we have $2\#_a(I) > 2f$ and consequently $\#_a(I) > f$. □

THEOREM 6.4. *Let $n > 3f$. The set of f-acceptable conditions is not union-closed.*

PROOF. Let us assume that f-acceptable conditions can be composed with union. It then follows (due to Lemma 6.3) that consensus is solvable for all $I \in \mathcal{V}^n$, with $\mathcal{V} = \{a, b\}$, $n > 3$ and $f = 1$. But this is known to be impossible (FLP impossibility result [Fischer et al. 1985]). □

7. *Maximality of $C1'$ and $C2$*

In this section, we prove that $C1'$ (the refined version of $C1$) and $C2$ are maximal in the sense that consensus is not f-fault tolerant solvable when adding to them any new input vector I. By Theorem 5.7, this notion of maximality can be equivalently stated in terms of legality, and also in terms of acceptability. Here is the statement in terms of acceptability:

Definition 7.1. Given an f-acceptable condition C, we say that C is *maximal* if any vector added to C makes it non f-acceptable.

Notice that this definition makes sense because any subset of an acceptable condition is also acceptable.

7.1. MAXIMALITY OF $C1'$.

In Section 6.1, we defined $C1$, and, in Section 6.2, we presented a refined version of $C1$, $C1'$ with $C1 \subset C1'$ and associated parameters $P1'$, $S1'$. Since both conditions are f-acceptable, $C1$ is not maximal. We show here that $C1'$ is maximal.

THEOREM 7.2. *Condition $C1'$ is maximal.*

PROOF. To prove that $C1'$ is maximal we first need to check that $C1'$ is f-acceptable. This has been argued in Section 6.2.

Now, to prove that $C1'$ is maximal, we may use (by Theorem 5.7) the legality version of Definition 7.1: we show that for any vector I, $I \notin C1'$, $C1' \cup \{I\}$ is not f-legal. We assume $f > 0$ because otherwise the theorem is trivially true: if $f = 0$ there is no I with $I \notin C1'$.

Consider the graph $Gin(C1', f)$. Since $C1'$ is f-legal, for each connected component of $Gin(C1', f)$ there is a value that appears in everyone of its vectors. In fact, there is exactly one such value. This follows directly from the easy to check fact that every connected component G_i of $Gin(C1', f)$ contains exactly one *corner vector* with all entries equal to the same value, that is, of the form a^n, for some value a. To see this, for any given $I \in C1'$, take an a such that $\#_a(I) = \#_{1st}(I)$ (which is unique since $f > 0$). Then there is a path in $Gin(C1', f)$ from I to a^n, which can be obtained by switching one at a time each one of the entries of I different from a to a. This means that the only value in common to all vectors of the connected component G_i that contains I is a, the most common value of I. Indeed, this is consistent with the fact that $S1(I) = a \ : \ \#_a(I) = \#_{1st}(I)$.

The technique to prove that $C1' \cup \{I\}$, $I \notin C1'$, is not f-legal consists of showing that I has edges to two different connected components of $Gin(C1', f)$. Hence, these two connected components belong to the same connected component of $Gin(C1' \cup \{I\}, f)$, which implies that $C1' \cup \{I\}$ is not f-legal (because two corner vectors cannot be in the same connected component of a legal condition).

That is, we show that I differs in at most f entries with vectors $I_1, I_2 \in C1'$ such that their most common input values are different, $S1'(I_1) \neq S1'(I_2)$.

Notice that $\#_{1st}(I) - \#_{2nd}(I) \leq f$, because $I \notin C1'$. Also, I contains at least two different values, because $I \notin C1'$ (i.e., every corner vector is in $C1'$). Let us denote by a the value such that $\#_a(I) = \#_{1st}(I)$, and by b the smallest value such that $\#_b(I) = \#_{2nd}(I)$. We consider two cases.

—Let us first assume that I is such that $\#_{1st}(I) - \#_{2nd}(I) = f$. As $I \notin C1'$, it is not the case that $\forall b : \#_b(I) = \#_{2nd}(I)$ it holds $a < b$. Thus, b is the smallest value such that $\#_b(I) = \#_{2nd}(I)$, and we we have that $b < a$.
  –Let I_1 be the vector obtained from I by switching one entry different from a to a. We have that $\#_a(I_1) = \#_a(I) + 1$, and hence $I_1 \in C1'$ with $S1'(I_1) = a$.
  –On the other hand, let I_2 be the vector obtained from I by switching f entries from a to b (notice that those f entries equal to a do exist). It follows that $\#_b(I_2) = \#_{1st}(I_2)$ and $\#_{1st}(I_2) - \#_{2nd}(I_2) \geq f$, with b being smaller than those appearing second (if any) in I_2. Hence, $I_2 \in C1'$ with $S1'(I_2) = b$.

—Let us now assume that I is such that $\#_{1st}(I) - \#_{2nd}(I) < f$. Since I contains at least two different values, we have $1 \leq \#_{1st}(I) - \#_{2nd}(I)$.
  –If $x = n - \#_{1st}(I)$, let I_1 be the vector obtained from I by switching $\min(x, f)$ entries different from a to a. Thus, I_1 differs from I in at most f entries. Also, $S1'(I_1) = a$, because either I includes f entries different from a, and $\#_{1st}(I_1) - \#_{2nd}(I_1) > f$ (since $1 \leq \#_{1st}(I) - \#_{2nd}(I)$), or else I_1 includes only a's.
  –Let I_2 be the vector obtained from I by switching f entries different from b to b, choosing first entries equal to a. That is, if there are f entries equal to a, switch them to b; else, switch all entries equal to a to b and then switch other entries different from b to b until we reach f switched entries or all entries become equal to b. Thus, I_2 differs from I in at most f entries. To prove that $S1'(I_2) = b$, observe that either (i) all entries from I_2 are equal to b or else (ii) $\#_{1st}(I_2) - \#_{2nd}(I_2) \geq f$. In case (i), clearly $S1'(I_2) = b$ and we are done. For case (ii), there are two subcases; it is clear that in both cases b is the single value that appears the most often in I_2:

    (ii.a) $\#_{2nd}(I_2) = \#_a(I_2)$, i.e. a is a value that appears the second most often in I_2. In this case, from $\#_a(I) - \#_b(I) < f$, $\#_a(I_2) = \max(\#_a(I) - f, 0)$ and $\#_b(I_2) = \min(\#_b(I) + f, n)$, we get $\#_b(I_2) - \#_a(I_2) \geq 2f - (\#_a(I) - \#_b(I)) \geq f + 1$. If follows $I_2 \in C1 (\subset C1')$, and consequently, $S1'(I_2) = b$.

    (ii.b) a is not a value that appears the second most often in I_2. Let c be such a value, $\#_{2nd}(I_2) = \#_c(I_2)$. In this case, we have $\#_b(I_2) - \#_c(I_2) \geq f$. Recall that b was the smallest value that appears the second most often in I. Hence $b < c$. We conclude from $\#_b(I_2) - \#_c(I_2) \geq f$ and $b < c$, that I_2 belongs to $C1'$ and $S1'(I_2) = b$. □

7.2. MAXIMALITY OF $C2$. Recall that $C2$, $P2$ and $S2$ are defined as follows:

—$I \in C2$ iff $\#_{\max(I)}(I) > f$.
—$P2(J) \equiv \#_{\max(J)}(J) > f - \#_{\perp}(J)$.
—$S2(J) = \max(J)$, the largest non-$\perp$ value of J.

THEOREM 7.3. *Condition $C2$ is maximal.*

PROOF. The structure of the proof is similar to the proof of Theorem 7.2. To prove that $C2$ is maximal, we first need to check that $C2$ is f-acceptable. This has been argued in Theorem 6.2.

Now, to prove that $C2$ is maximal we may use (by Theorem 5.7) the legality version of Definition 7.1: we show that for any vector I, $I \notin C2$, $C2 \cup \{I\}$ is not f-legal. We assume $f > 0$ because otherwise the theorem is trivially true: if $f = 0$ there is no I with $I \notin C2$.

Consider the graph $Gin(C2, f)$. Since $C2$ is f-legal, for each connected component of $Gin(C2, f)$ there is a value that appears in everyone of its vectors. In fact, there is exactly one such value. This follows directly from the easy to check fact that every connected component G_i of $Gin(C2, f)$ contains exactly one *corner vector* with all entries equal to the same value, i.e., of the form a^n, for some value a. Moreover, the only value in common to all vectors of the connected component G_i that contains I is a, the largest value of I. Indeed, this is consistent with the fact that $S2(I) = \max(I)$.

We now show that if $I \notin C2$, then $C2 \cup \{I\}$ is not f-legal. As in Theorem 7.2, we prove this by showing that I differs in at most f entries with vectors $I_1, I_2 \in C2$ such that $S2(I_1) \neq S2(I_2)$, that is, in different connected components of $Gin(C2, f)$, and hence these two components are joined in $Gin(C2 \cup \{I\}, f)$. A contradiction because the new connected component of $Gin(C2 \cup \{I\}, f)$ has two different corner vectors, and hence $Gin(C2 \cup \{I\}, f)$ is not f-legal. This is done as follows (this part of the proof is different from the corresponding part in the proof for $C1'$).

Notice that, if a is the largest value of I, then $\#_a(I) \leq f$, since $I \notin C2$ (and this makes sense because we assumed $f > 0$). Let b be the second largest value of I (which exists since $f < n$). Let y be the number of entries of I that are different from a. Due to $f < n$ and $y + \#_a(I) = n$, we conclude that $y \geq f - \#_a(I) + 1$. Let us now define I_1 as the vector obtained from I by switching $f - \#_a(I) + 1$ entries different from a to a (due to the previous observation on y, those entries do exist). Moreover, as $\#_a(I) \geq 1$, we have $f - \#_a(I) + 1 \leq f$, hence $I1$ differs from I in at most f entries. We have $\#_a(I_1) = \#_a(I) + f - \#_a(I) + 1 = f + 1$, and hence $I_1 \in C2$ with $S2(I_1) = a$. On the other hand, let I_2 be the vector obtained from I by switching first all entries equal to a to b (the number of these entries is $\leq f$), and then others until a total of $f - \#_b(I) + 1$ entries different from b are changed to b. As $\#_b(I) \geq 1$, it follows that $f - \#_b(I) + 1 \leq f$. Hence, $I2$ differs from I in at most f entries. We have that $\#_a(I_2) = 0$, and $\#_b(I_2) = \#_b(I) + f - \#_b(I) + 1 = f + 1$ with b being the largest such value. It follows that $I_2 \in C2$ with $S2(I_2) = b$. □

8. *The Condition-Based Approach in Message Passing Systems*

The condition-based consensus protocol presented in Figure 2 uses single-writer multi-reader registers, and snapshot registers. The snapshot registers can be (wait-free) implemented in terms of single-writer multi-reader registers using the techniques of Afek et al. [1993], and hence the algorithm can be rewritten using only read/write registers. The resulting algorithm would be less efficient, since the simulation of each write or snapshot operation to the snapshot registers requires $O(n^2)$ read and write to read/write registers. The simulation of Attiya and Rachman [1998] is more efficient, but still has an overhead: $O(n \log n)$ per operation to a snapshot register.

```
Function MP_Consensus(v_i)
(1)  broadcast VAL(v_i, i);
(2)  wait until (at least (n − f) VAL(−, −) messages have been delivered);
(3)  forall j ∈ [1 · · n] do if (VAL(v_j, j) has been delivered)
(4)        then V_i[j] ← v_j else V_i[j] ← ⊥ endif enddo;
(5)  if P(V_i) then w_i ← S(V_i) else w_i ← ⊤ endif;
(6)  UR_broadcast ECHO(v_i, w_i, i);
(7)  repeat wait until (a new ECHO(v_j, w_j, j) message has been delivered);
(8)        W_i[j] ← w_j; Y_i[j] ← v_j;
(9)        if (ECHO(−, w, −) messages with the same value w (≠ ⊥)
(10)          delivered from a majority of processes )
(11)                then return(w) endif
(12) until (⊥ ∉ W_i) endrepeat;
(13) return(F(Y_i))
```

FIG. 3. A message passing consensus protocol for $f < n/2$.

It is known [Attiya et al. 1995] that any wait-free algorithm in the read/write shared memory model which uses atomic, single-writer multi-reader registers can be executed in the message passing model when $f < n/2$ [Gafni 1998]. Moreover, there exists an efficient simulation [Attiya 2000], where each read or write operation is simulated with only $O(n)$ messages. Thus, our condition-based consensus protocol can be automatically transformed to solve the same problem in a message passing system when $f < n/2$, by first eliminating snapshot operations using the simulation of Attiya and Rachman [1998] and then simulating read/write operations with message passing using the algorithm of Attiya [2000], with an overhead of $O(n \log n)$ messages. This section discusses the design of a simple message passing condition-based consensus protocol, inspired by the protocol of Figure 2, but without the overhead of an automatic transformation. In addition, we show that there is no condition-based consensus protocol for message passing if $f \geq n/2$. Notice that the fact that the transformations mentioned above require $f < n/2$ (as does any general transformation) does not directly imply this result.

8.1. ADAPTING THE PROTOCOL TO THE MESSAGE PASSING CONTEXT. The main difficulty in adapting the protocol of Figure 2 to a message passing system comes from the absence of a message passing snapshot-like primitive providing the ordering property $(J1 \leq J2) \vee (J2 \leq J1)$ where $J1$ and $J2$ are the results of two invocations to the snapshot primitive.

Figure 3 describes a message passing protocol that solves consensus when $f < n/2$, for any f-acceptable condition C. This protocol is an adaptation of the Figure 2 protocol to the message passing model without simulating the snapshot operation, and instead, relaying on the "majority of correct processes" assumption. Its proof is very similar to the proof of the protocol described in Figure 2 and is omitted.

The protocol assumes *broadcast* and *uniform reliable broadcast* communication facilities. Both the invocations of broadcast(m) and UR_broadcast(m) entail the sending of the message m to all processes. The broadcast(m) primitive is not reliable in the following sense: if the sender crashes while it is broadcasting m, it is possible that only a subset of the processes receive and deliver m. The UR_broadcast(m) primitive is more reliable: if a process receives and delivers m, then all correct processes receive and deliver m. Both primitives can be implemented in the asynchronous

message passing model considered, although UR_broadcast is more costly (e.g., Hadzilacos and Toueg [1993]). UR_broadcast is needed to ensure that if a process terminates, then all correct processes do terminate.

8.2. ON THE PRESENCE OF AN ATOMIC SNAPSHOT PRIMITIVE. When $I \in C$, both protocols rely on $A_{P \to S}$ to ensure the agreement property. But, when $I \notin C$, they rely on different requirements. As shown in Lemma 4.6, the shared memory protocol relies on $A_{P \to S}$ to guarantee agreement. In contrast, the message passing protocol relies on the assumption $f < n/2$. Indeed, in the absence of a snapshot primitive, the following scenario is possible. Assume $I \notin C$ and p_i gets the local view $J1$ and $P(J1)$ is true; p_j gets the local view $J2$ and $P(J2)$ is true. As now it is possible that $\neg(J1 \leq J2) \wedge \neg(J2 \leq J1)$, it follows that $A_{P \to S}$ alone is insufficient to prevent to have $S(J1) \neq S(J2)$ (and guarantee agreement) when $I \notin C$. Hence, the majority of correct processes requirement. Notice that the construction of a snapshot primitive in a message passing system also requires the $f < n/2$ assumption.[6]

As we show in the next theorem, there is no consensus protocol if $f < n/2$ does not hold. Interestingly, there *is* a message passing protocol that solves consensus for *any* value of f when the inputs do not violate the condition C. The protocol of Figure 3 without the majority requirement in line (8) does the job, since as discussed above, agreement in this case relays only on the condition $A_{P \to S}$. Thus, the difficulty of solving consensus when $f \geq n/2$ comes from the requirement to deal also with input vectors not in C.

A condition C is *f-non-trivial* if in its graph $Gin(C, f)$ (defined in Section 5.1), the intersection of the $d(G_i)$'s over all connected components G_i is empty. The intuition behind this definition is that if this intersection contains a value v, then a trivial solution to the consensus exists, where processes choose the default value v with no communication at all. The following stands in contrast to Theorem 4.2.

THEOREM 8.1. *Let C be an f-acceptable, f-non-trivial condition. There is no f-fault tolerant protocol that solves the consensus problem for C in a message passing system when $f \geq n/2$. If $f < n/2$, then there is an f-fault tolerant consensus protocol for any f-acceptable C.*

PROOF. The second part of the theorem, when $f < n/2$, follows from the correctness of the protocol of Figure 3.

For the first part, assume $f \geq n/2$. Assume for contradiction that such a protocol exists. Thus, as explained in the proof of Lemma 5.5, Theorem 5.4 implies that the protocol decides on one value, v_i, for each connected component G_i. First notice that there must exist two different connected components of $Gin(C, f)$, G_1, G_2, where the protocol decides v_1 with inputs of G_1, and v_2 with inputs of G_2, where $v_1 \neq v_2$. Otherwise, for every connected component G_i, the protocol decides the same value v, which by the validity condition has to be in common to every vector of every G_i, contradicting the assumption that C is f-non-trivial.

Consider vectors I_1, I_2 of G_1, G_2 respectively. Let $p_1, \ldots, p_{\lfloor n/2 \rfloor}$ run starting on I_1 until they decide, without hearing any messages from the other processes. They

[6] A more constraining result, namely, show that there is no simulation of a read/write atomic register when $f \geq n/2$, is the topic of the exercise 10.15 in Attiya and Welch [1998].

have to decide eventually, because it is possible that the other processes crash from the very beginning ($f > n/2$), and because I_1 is in C. Moreover, the decision must be v_1, since I_1 is in G_1.

Consider the prefix of this (infinite) execution, until the point they decide, call it α_1, and do not deliver any message from $p_1, \ldots, p_{\lfloor n/2 \rfloor}$ to the others. Do the same for I_2, running only the other processors, $p_{\lfloor n/2 \rfloor + 1}, \ldots, p_n$, and call the prefix until they decide v_2, α_2.

Consider the input vector I constructed with the first $\lfloor n/2 \rfloor$ entries from I_1, and the other entries from I_2 (I is not necessarily in C). Construct the execution which starts in I by pasting α_1 first, then α_2, and then delivering all messages between the two groups. This execution violates agreement. □

8.3. COMING BACK TO THE SHARED MEMORY MODEL. Here we discuss a shared memory protocol inspired by the previous message passing protocol that does not use snapshots, but works only for $f < n/2$.

The message-passing protocol described in Figure 3 can be translated to get a shared memory protocol that does not use snapshot operations. The shared memory protocol we obtain (let's call it A) differs from the protocol described in Figure 2 (let's call it B) mainly in two points:

—A requires $f < n/2$, while B assumes $f < n$.

—Due to the use of the snapshot operation, the local views obtained by the processes are ordered by containments in B. This containment property is not provided in A that basically uses a collect operation.[7] It follows that, while both A and B satisfy the Validity, Agreement and Guaranteed_Termination properties, B terminates in more cases than A when the actual input vector $J \notin \mathcal{C}_f$. To illustrate this point let us consider the case where the input vector $J \notin \mathcal{C}_f$, and there is a single process p_j that gets a local view J_j such that $P(J_j)$ holds. Moreover, p_j writes $w_j \neq \top$ in the shared memory and then crashes.

  —In B, the correct processes terminate (see Theorem 4.8).

  —In A, no correct process terminates. This is due to the fact that a majority of processes have to suggest the same non-$\perp$ value before deciding it.

9. *Trading Safety for Liveness*

9.1. NONTERMINATION VS. TERMINATION. As far as termination is concerned, we have seen that the protocol described in Figure 2 satisfies Guaranteed_Termination (defined in Section 3.1) and P-More_Termination (defined in Theorem 4.8). Hence, a correct process fails to decide only in rare situations.

It is actually possible to modify the protocol to guarantee that a correct process always terminates provided there are no more than f crashes. This has a price that translates as a versatile tradeoff between the liveness and the safety guaranteed by the protocol. More precisely, the safety is weakened in the sense that the default value $\perp$ can now be decided in some circumstances in order to prevent nontermination [Jayanti et al. 1998; Raynal 1997].

[7] collect is an operation that non-atomically executes a set of read operations: collect(X) is equivalent to **forall** j: **do** read($X[j]$) **enddo**.

```
Function SM_Term_Consensus(v_i)
(1)  write(v_i, V[i]);
(2)  repeat V_i ← snapshot(V) until (#(V_i[j] ≠ ⊥) ≥ (n − f)) endrepeat;
(3)  if P(V_i) then w_i ← S(V_i) else w_i ← ⊤ endif;
(4)  write(w_i, W[i]);
(5)  r_i ← 0;
(6)  repeat forall j ∈ [1 · · n] do W_i[j] ← read(W[j]) enddo;
(7)       if (∃ j : W_i[j] ≠ ⊥, ⊤) then return(W_i[j]) endif;
(8)       r_i ← r_i + 1
(9)  until ((r_i = N) ∨ (⊥ ∉ W_i))  endrepeat;
(10) if (⊥ ∉ W_i) then forall j ∈ [1 · · n] do Y_i[j] ← read(V[j]) enddo;
(11)                   return(F(Y_i))
(12)              else return(⊥)
(13) endif
```

FIG. 4. A terminating condition-based consensus protocol.

9.2. A TERMINATING PROTOCOL. Given a condition C and assuming at most f crashes, a "terminating" protocol is described in Figure 4. The local variable r_i is a counter used to limit the number of iterations executed by p_i. $N \in [1 \cdot\cdot \infty)$ is a tuning parameter for the trading of safety for liveness; $N = +\infty$ corresponds to the no-trading case where the value $\perp$ can never be decided (Figure 2). This protocol guarantees the following properties (their proofs are left to the reader):

—T-Validity: A decided value is a proposed value or $\perp$.

—T-Obligation: If the input vector satisfies C, then $\perp$ cannot be decided.

—T-Agreement: No two processes decide different proposed values.

—T-Termination: When at most f processes crash, every correct process decides.

Let us note that, when the input vector I does not satisfy C, it is possible that some processes decide a proposed value while others decide $\perp$.

Let us consider the property $T_{C \to P}$ defined in Section 4. This property has been used in Lemma 4.3 (Section 4.3) to prove the consensus termination property of the generic protocol described in Figure 2. In contrast, the termination of the protocol described in Figure 4 relies only on the assumption that there are no more than f crashes. As far as the property $T_{C \to P}$ is concerned, it is still necessary but for another purpose: the protocol requires it in order to satisfy the obligation property. This interesting feature constitutes another facet of the tradeoff relating the safety and the liveness properties guaranteed by the protocol of Figure 4.

As before, this protocol can be easily adapted to the message passing context without snapshot primitive, but with the $f < n/2$ assumption (see Figure 5). Interestingly, the uniform reliable broadcast is no longer necessary.

10. *Conclusion*

Solving the consensus problem consists of providing each process with the same view of a relevant part of the system. The difficulty of solving this problem in an asynchronous system prone to process crash failures comes from the uncertainty created by asynchrony, failures and ignorance of the actual input values proposed by the processes in the execution [Lynch 1989]. This article has investigated a new

```
Function MP_Term_Consensus(v_i)
(1) broadcast VAL(v_i, i);
(2) wait until (at least (n − f) VAL(−, −) messages have been delivered);
(3) forall j ∈ [1 · · n] do if (VAL(v_j, j) has been delivered)
(4)         then V_i[j] ← v_j else V_i[j] ← ⊥ endif enddo;
(5) if P(V_i) then w_i ← S(V_i) else w_i ← ⊤ endif;
(6) broadcast ECHO(w_i);
(7) wait until (ECHO messages have been delivered from at least a majority of processes);
(8) if (the same value d ≠ ⊤ appears in a majority of ECHO messages)
(9)         then return(d) else return(⊥) endif
```

FIG. 5. A terminating message passing consensus protocol.

condition-based approach, to cope with these difficulties. Two main results have been presented. The first is a generic consensus protocol for the shared memory model. A set of *acceptable* conditions has been defined, and shown that for any such condition the protocol has the following noteworthy properties: it always guarantees agreement and validity, and it terminates (at least) when the inputs satisfy the condition with which the protocol has been instantiated, or when there is no crash. The second main result of the article is the statement of a general characterization that captures the set of all the conditions allowing a consensus solution. These are exactly the acceptable conditions. Thus, the characterization states the minimal properties a condition has to satisfy for consensus to be solvable.

The article presented several additional results. It described two natural conditions and proved them to be acceptable. Furthermore, it showed that these conditions cannot be extended and remain acceptable. The article has also presented an efficient version of the generic protocol for the message passing model. It has also shown how the protocol safety can be traded for liveness.

An interesting line of research is trying to combine the condition based approach with other approaches that have been proposed to circumvent the FLP impossibility result, like failure detectors, randomization, or communication objects stronger than read/write registers. We describe some preliminary results in this direction in Mostefaoui et al. [2002] for failure detectors and randomization.

We have defined a version of the consensus problem for the condition based approach in terms of three requirements. The first two are the agreement and validity requirements of the classic consensus problem, and are independent of a particular condition C. The third requirement, requires termination under "normal" operating scenarios, including (1) inputs belonging to C, and (2) failure-free executions. Part (1), requires termination even in executions where some processes crash initially and their inputs are unknown to the other processes. This is represented by a view J with $\perp$ entries for those processes. Termination is required if it is possible that the full input vector belongs to C, that is, if J can be extended to an input vector $I \in C$. Part (2) defines two well-behaved scenarios where a protocol should terminate even if the input vector does not belong to C. In Section 4.4, we show that our protocol terminates in other situations as well. It would be interesting to understand better other "normal" termination conditions for input vectors not in the condition, including randomized conditions that require processes to terminate with high probability when the inputs do not satisfy the condition.

We remark that the set of acceptable conditions is quite rich. In a sequel article [Mostefaoui et al. 2001a], we describe one way of producing acceptable conditions

using a weight function. Also, acceptable conditions, in general, do not satisfy the closure properties needed for the BG-simulation [Borowsky et al. 2001] to work. This simulation shows how to transform a protocol that solves a problem with some resilience f into a protocol that solves the same problem with a different resilience f', $f < f'$. However, for the simulation to work, the version of the problem for f and the version of the problem for f' have to be related in some specific way. Thus, if an acceptable condition $\mathcal{C}$ for f and an acceptable condition $\mathcal{C}'$ for f', satisfy this relationship we can derive results from one level of resilience to another, and, for example, study its wait-free solvability to derive the solvability of the condition for general f.

In this article, we have proved that acceptable conditions are exactly the class of conditions that allow consensus to be solved. It may be that among the acceptable conditions, in some sense, some are easier than others. In Mostefaoui et al. [2001b], we have presented a hierarchy of acceptable conditions, and presented a protocol whose time complexity depends on the place in the hierarchy of its condition. However, no lower bounds are known. Finally, the interested reader can find a probabilistic evaluation of the condition-based approach in Mostefaoui et al. [2003].

ACKNOWLEDGMENTS. We are grateful for the many comments of anonymous referees that helped improve the article significantly.

REFERENCES

AFEK, Y., ATTIYA, H., DOLEV, D., GAFNI, E., MERRITT, M., AND SHAVIT, N. 1993. Atomic snapshots of shared memory. *J. ACM 40*, 4, 873–890.

AGUILERA, M., AND TOUEG, S. 1998. Failure detection and randomization: a hybrid approach to solve consensus. *SIAM J. Comput. 28*, 3, 890–903.

ASPNES, J. 2000. Fast deterministic consensus in a noisy environment. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing* (*PODC'00*) (Portland, Calif.). ACM, New York, 299–308.

ATTIYA, H., AND AVIDOR, Z. 2002. Wait-free n-set consensus when inputs are restricted. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC'02)* (Toulouse, France). Number 2508. Springer-Verlag, New York, 326–338.

ATTIYA, H. 2000. Efficient and robust sharing of memory in message passing systems. *J. Algorithms 34*, 1, 109–127.

ATTIYA, H., BAR-NOY, A., AND DOLEV, D. 1995. Sharing memory robustly in message passing systems. *J. ACM 42*, 1, 124–142.

ATTIYA, H., AND RACHMAN, O. 1998. Atomic snapshots in o(n log n) operations. *SIAM J. Comput. 27*, 2, 319–340.

ATTIYA, H., AND WELCH, J. 1998. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. McGraw–Hill, New York.

AUMANN, Y. 1997. Efficient asynchronous consensus with the weak adversary scheduler. In *16th ACM Symposium on Principles of Distributed Computing (PODC'97)* (Santa Barbara, Calif.). ACM, New York, 209–218.

AZAR, Y., BRODER, M., AND MANASSE, M. 1993. On-line choice of on-line algorithms. In *Proceedings of the 4th Annual ACM/SIAM Symposium on Discrete Algorithms*. ACM, New York, 432–440.

BEN-OR, M. 1983. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing (PODC'83)*. ACM Press, Montreal, 27–30.

BERMAN, P., AND GARAY, J. 1998. Adaptability and the usefulness of hints. In *Proceedings of the 6th European Symposium on Algorithms (ESA'98)*. Number 2508. Springer-Verlag, New York, 271–282.

BIRAN, O., MORAN, S., AND ZAKS, S. 1990a. A combinatorial characterization of the distributed 1-solvable tasks. *J. Algorithm 11*, 420–440.

BIRAN, O., MORAN, S., AND ZAKS, S. 1990b. Deciding 1-solvability of distributed tasks is np-hard. In *Proceedings of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'90)*. Number 484. Springer-Verlag, New York, 206–220.

BOROWSKY, E., GAFNI, E., LYNCH, N., AND RAJSBAUM, S. 2001. The BG distributed simulation algorithm. *Dist. Comput. 14*, 13, 127–146.

CASTRO, E.-V., AND WOOD, D. 1992. A survey of adaptive sorting algorithms. *ACM Comput. Surv. 24*, 4, 441–476.

CHANDRA, T., HADZILACOS, V., AND TOUEG, S. 1996. The weakest failure detector for solving consensus. *J. ACM 43*, 4, 685–722.

CHANDRA, T., AND TOUEG, S. 1996. Unreliable failure detectors for reliable distributed systems. *J. ACM 43*, 2, 225–267.

CHAUDHURI, S. 1993. More choices allow more faults: set consensus problems in totally asynchronous systems. *Inf. and Comput. 105*, 132–158.

DOLEV, D., DWORK, C., AND STOCKMEYER, L. 1987. On the minimal synchronism needed for distributed consensus. *J. ACM 34*, 1, 77–97.

DOLEV, D., LYNCH, N., PINTER, S., STARK, E., AND WEIHL, W. 1986. Reaching approximate agreement in the presence of faults. *J. ACM 33*, 3, 499–516.

DWORK, C., LYNCH, N., AND STOCKMEYER, L. 1988. Consensus in the presence of partial synchrony. *J. ACM 35*, 2, 288–323.

FISCHER, M., LYNCH, N., AND PATERSON, M. 1985. Impossibility of distributed consensus with one faulty process. *J. ACM 32*, 2, 374–382.

FRIEDMAN, R., MOSTEFAOUI, A., RAJSBAUM, S., AND RAYNAL, M. 2002. Asynchronous distributed agreement and its relation with error correcting codes. In *Proceedings of the 16th International Symposium on DIStributed Computing (DISC'02)* (Toulouse, France). Number 2508. Springer-Verlag, New York, 63–87.

GAFNI, E. 1998. *Distributed Computing: a Glimmer of a Theory, in Handbook of Computer Science*. CRC Press.

GAFNI, E., AND KOUTSOUPIAS, E. 1999. Three-processor tasks are undecidable. *SIAM J. Comput. 28*, 3, 970–983.

GUERRAOUI, R. 1995. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG)*. Number 972. Springer-Verlag, New York, 87–100.

GUERRAOUI, R., AND RAYNAL, M. 2003. A generic framework for indulgent consensus. In *Proceedings of the 23th IEEE International Conference on Distributed Computing Systems (ICDCS'03)* (Providence, R.I.). IEEE Computer Society Press, Los Alamitos, Calif., 88–97.

HADZILACOS, V., AND TOUEG, S. 1993. *Reliable Broadcast and Related Problems, in Distributed Systems*. S. Mullender, Ed. ACM Press, New York.

HERLIHY, M. 1991. Wait-free synchronization. *ACM Trans. Prog. Lang. Syst. 11*, 1, 124–149.

HERLIHY, M., AND RAJSBAUM, S. 1997. On the decidability of distributed decision tasks. In *Proceedings of the 29th ACM Symposium on the Theory of Computing (STOC'97)*. ACM, New York, 589–598.

HERLIHY, M., AND RAJSBAUM, S. 1999. New perspectives in distributed computing. In *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science (MFCS'99)*. Number 1672. Springer-Verlag, New York, 170–186. (Invited Talk).

HERLIHY, M., AND SHAVIT, N. 1999. The topological structure of asynchronous computability. *J. ACM 46*, 6, 858–923.

HERLIHY, M., AND WING, J. 1990. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Prog. Lang. Syst. 12*, 3, 463–492.

JAYANTI, P., CHANDRA, T., AND TOUEG, S. 1998. Fault-tolerant wait-free shared objects. *J. ACM 45*, 3, 451–500.

LAMPORT, L. 1998. The part-time parliament. *ACM Trans. Comput. Syst. 16*, 2, 133–169.

LOUI, M., AND ABU-AMARA, H. 1987. *Memory Requirements for Agreement Among Unreliable Asynchronous Processes, in Parallel and Distributed Computing* (Greenwich, Conn.). Advances in Computing Research, vol. 4. JAI Press.

LYNCH, N. 1989. A hundred impossibility proofs for distributed computing. In *Proceedings of the 8th ACM Symposium on Principles of Distributed Computing (PODC'89)* (Edmonton, Ont., Canada). ACM, New York, 1–27. (Invited Talk).

LYNCH, N. 1996. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, Calif.

MORAN, S., AND WOLFSTAHL, Y. 1987. Extended impossibility results for asynchronous complete networks. *Inf. Proc. Lett. 26*, 145–151.

MOSES, Y., AND RAJSBAUM, S. 2002. A layered analysis of consensus. *SIAM J. Comput. 31*, 4, 989–1021.

MOSTEFAOUI, A., MOURGAYA, E., RAIPIN, P., AND RAYNAL, M. 2003. Evaluating the condition-based approach to solve consensus. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN03)* (San Francisco, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., 541–550.

MOSTEFAOUI, A., RAJSBAUM, S., AND RAYNAL, M. 2001. Conditions on input vectors for consensus solvability in asynchronous distributed systems. In *Proceedings of the 33rd ACM Symposium on Theory of Computing (STOC'01)* (Crete, Greece). ACM Press, New York, 153–162.

MOSTEFAOUI, A., RAJSBAUM, S., AND RAYNAL, M. 2002. A versatile and modular consensus protocol. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'02)* (Washington, D.C.). IEEE Computer Society Press, Los Alamitos, Calif., 364–373.

MOSTEFAOUI, A., RAJSBAUM, S., AND RAYNAL, M. 2003. Using conditions to expedite consensus in synchronous distributed systems. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC'03)* (Sorrento, Italy). Lecture Notes in Computer Science, vol. 2848. Springer-Verlag, New York, 249–263.

MOSTEFAOUI, A., RAJSBAUM, S., RAYNAL, M., AND ROY, M. 2001a. Efficient condition-based consensus. In *Proceedings of the 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO 8)* (Catalonia, Spain). Carleton Univ. Press, 275–292.

MOSTEFAOUI, A., RAJSBAUM, S., RAYNAL, M., AND ROY, M. 2001b. A hierarchy of conditions for consensus solvability. In *Proceedings of the 20th ACM SIGACT-SIGOPS International Symposium on Principles of Distributed Computing (PODC'01)* (Newport, R.I.). ACM, New York.

MOSTEFAOUI, A., RAJSBAUM, S., RAYNAL, M., AND ROY, M. 2002. Condition-based protocols for set agreement problems. In *Proceedings of the 16th International Symposium on Distributed Computing (DISC'02)* (Toulouse, France). Lecture Notes in Computer Science, vol. 2508. Springer-Verlag, New York, 48–62.

MOSTEFAOUI, A., AND RAYNAL, M. 1999. Solving consensus using Chandra-Toueg-s unreliable failure detectors: a general quorum-based approach. In *Proceedings of the 13th International Symposium on Distributed Computing (DISC'99)* (Bratislava), P. Jayanti, Ed. Lecture Notes in Computer Science, vol. 1603. Springer-Verlag, New York, 49–63.

MOSTEFAOUI, A., RAYNAL, M., AND TRONEL, F. 2000. The best of both worlds: A hybrid approach to solve consensus. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'00)* (New York, N.Y.). IEEE Computer Society Press, Los Alamitos, Calif., 513–522.

RAYNAL, M. 1997. Real-time dependable decisions in timed asynchronous distributed systems. In *Proceedings of the 3rd International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'97)* (Newport Beach, Calif.). IEEE Computer Society Press, Los Alamitos, Calif., 283–290.

TAUBENFELD, G., KATZ, S., AND MORAN, S. 1994. Impossibility results in the presence of multiple faulty processes. *Inf. Comput. 113*, 2, 173–198.

TAUBENFELD, G., AND MORAN, S. 1996. Possibility and impossibility results in a shared memory environment. *Acta Inf. 35*, 1–20.

ZIBIN, Y. 2003. Condition-based consensus in synchronous systems. In *Proceedings of the 17th International Symposium on Distributed Computing (DISC'03)* (Sorrento, Italy). Lecture Notes in Computer Science, vol. 2848. Springer-Verlag, New York, 239–248.

RECEIVED JANUARY 2002; REVISED JUNE 2003; ACCEPTED JULY 2003