

# Condition Adaptation in Synchronous Consensus

Taisuke IZUMI      Toshimitsu MASUZAWA

Graduate School of Information Science and Technology, Osaka University  
{t-izumi, masuzawa}@ist.osaka-u.ac.jp

## Contact Author

Taisuke IZUMI

Graduate School of Information Science and Technology, Osaka University  
1-3 Machikaneyama, Toyonaka, 560-8531, Japan  
+81-6-6850-6584  
t-izumi@ist.es.osaka-u.ac.jp

## Abstract

The condition-based approach is one of sophisticated methods to overcome several impossibility results in distributed consensus problem (e.g., impossibility of fault tolerance in asynchronous consensus, or time complexity lower bounds in synchronous consensus). It introduces conditions on input vectors to specify subsets of all possible input vectors to consensus algorithms, and condition-based algorithms can circumvent the impossibility if actual input vectors satisfy a particular condition. In this paper, we present a new condition-based paradigm for synchronous consensus. We newly introduce the concept of adaptation on the time complexity of condition-based algorithms, and present the adaptive condition-based approach to synchronous consensus. In our approach, all possible input vectors are classified into hierarchical conditions according to their difficulty called the legality level. Execution time of adaptive condition-based algorithms depends on the legality level of input vectors.

We propose two adaptive condition-based algorithms for synchronous consensus. The first algorithm requires that the majority of processes are correct, and terminates within  $\min\{f + 2, t + 1\} - l$  rounds if  $l < f$ , where  $f$  and  $t$  is the actual and the maximum numbers of faults respectively, and  $l$  is the legality level of the input vector. Moreover, the algorithm terminates in one round if  $l \geq t$  and  $f = 0$ , and terminates within two rounds if  $l \geq f$  holds. Compared with previous algorithms, this algorithm achieves the best time complexity. The second algorithm can tolerate any number of faults, and terminates within  $\max\{3, \min\{f + 3, t + 2\} - l\}$  rounds if  $l < f$  holds, terminates in one round if  $l \geq t$  and  $f = 0$ , and terminates within three rounds if  $l \geq f$  holds.

**Index Terms** : distributed algorithm, synchronous system, consensus problem, fault-tolerance, crash fault, condition-based approach, adaptive algorithm

# 1 Introduction

## 1.1 Condition-Based Approach

The *uniform consensus* problem is a fundamental and important problem for designing fault-tolerant distributed systems. Informally, the uniform consensus problem is defined as follows: each process proposes a value, and all non-faulty processes have to agree on a common value that is proposed by a process. The uniform consensus problem has many applications, e.g., atomic broadcast [5, 13], shared objects [2, 14], weak atomic commitment [11] and so on. However, despite of the variety of its applications, the uniform consensus problem is known to be unsolvable by deterministic solution in asynchronous systems subject to only a single crash fault [9]. Thus, several approaches to circumvent this impossibility, such as synchrony [7], randomization [3] and unreliable failure detectors [5], have been proposed.

As one of such approaches, the *condition-based approach* is recently introduced [16]. The principle of this approach is to restrict inputs so that the generally-unsolvable problem can become solvable. A *condition* represents some restriction to inputs. In the case of the uniform consensus problem, it is defined as a subset of all possible *input vectors* whose entries correspond to the proposal of each process. The first result of the condition-based approach clarifies the condition for which the uniform consensus can be solved in asynchronous systems subject to crash faults [16]. More precisely, this result presented a class of conditions, called *d-legal conditions*, and proved that the *d-legal conditions* is the class of necessary and sufficient conditions that make the uniform consensus solvable in asynchronous systems where at most *d* processes can crash.

Several following researches investigate the condition-based uniform consensus in synchronous systems. However, it is well-known that the uniform consensus problem can be solved in synchronous systems. Thus, these researches focus on improvement of the time complexity for the restricted input vectors. While it is known that any synchronous uniform consensus algorithm needs at least  $\min\{f + 2, t + 1\}$  rounds for termination, where *f* and *t* is the actual and the maximum numbers of faults respectively [6][8], more efficient algorithms can be realized if the condition-based approach is introduced. For example, the algorithm proposed by Mostefaoui et al. terminates within  $\min\{f + 2, t + 1 - d\}$  rounds if the input vector is in some *d-legal condition*, and terminates within  $\min\{f + 2, t + 1\}$  rounds otherwise [18]. To the best of our knowledge, three synchronous condition-based uniform consensus algorithms have been proposed [17, 18, 25].

## 1.2 Our Contribution

We also investigate condition-based uniform consensus algorithms in synchronous systems. In this paper, we advance the condition-based approach by newly introducing the concept of adaptation on the time complexity of condition-based algorithms. Intuitively, the adaptation in condition-based algorithm is the property that the execution time of algorithms depends on actual difficulty of input vectors. The previous researches [17, 18, 25] show that uniform consensus algorithms can terminate early by at most *d* rounds when the input vector is in some *d-legal condition*. In this sense, the value *d* can be regarded as difficulty of input vectors in *d-legal conditions*. However, it is also shown that a *d-legal condition* can contain a  $(d + 1)$ -legal condition as a subset [21]. This implies that a *d-legal condition* can include inputs vector whose difficulty is lower than *d*. Our adaptation concept guarantees that such easier input makes our algorithms terminate earlier. To explain the adaptation more precisely, we present an example for the *d-legal condition*  $C_d^{\max}$ : The

condition  $C_d^{\max}$  consists of the vectors in which the largest value in the vector appears more than  $d$  times. From the result in [18], we can construct, for any fixed  $d$ , an efficient condition-based uniform consensus algorithm that terminates within  $t + 1 - d$  rounds for any input vector in  $C_d^{\max}$ . Now let  $\mathcal{A}$  be such an algorithm for  $d = 2$ , and consider the three vectors  $I_1 = \langle 0, 1, 1, 1, 1 \rangle$ ,  $I_2 = \langle 0, 1, 2, 2, 2 \rangle$ , and  $I_3 = \langle 0, 1, 2, 3, 3 \rangle$ . Clearly, the vectors  $I_1$ , and  $I_2$  are in  $C_2^{\max}$ , and thus for the input vectors  $I_1$  and  $I_2$ , the algorithm  $\mathcal{A}$  terminates within  $t - 1$  rounds. On the other hand, since  $I_3$  is not in  $C_2^{\max}$ , the algorithm  $\mathcal{A}$  terminates within  $\min\{f + 2, t + 1\}$  rounds. However, from the definition,  $I_1$  is contained in  $C_3^{\max}$ , and  $I_3$  is contained in  $C_1^{\max}$ . Therefore, the execution for  $I_1$  and  $I_3$  is expected to terminate within  $t - 2$  rounds and  $t$  rounds respectively: This is what we call adaptation. However, in this sense, none of existing algorithms is adaptive.

This paper formalizes the adaptive condition-based approach, and proposes two adaptive condition-based uniform consensus algorithms, called **FACC** (Fast Adaptive Condition-based Consensus) and **ACC-ANF** (Adaptive Condition-based Consensus for Any Number of Faults). To define actual difficulty of input vectors, we introduce the notion of *legal condition sequence* and *legality level*. Intuitively, the legal condition sequence is a hierarchical sequence of  $d$ -legal conditions. The legality level is defined for a legal condition sequence, and represents the location of input vectors in the hierarchy<sup>1</sup>. In the previous example, the legal condition sequence is  $\langle C_0^{\max}, C_1^{\max}, \dots, C_5^{\max} \rangle$ , and the legality levels of  $I_1$ ,  $I_2$  and  $I_3$  are respectively 3, 2, and 1. Both of two proposed algorithms are instantiated by a legal condition sequence, and have adaptation to legality level. The algorithm **FACC** and **ACC-ANF** respectively have distinct advantages in the point of time complexity and fault resilience, and thus comparable with each other. The algorithm **FACC** assumes that majority of processes are correct. For any input vector with legality level  $l$ , it terminates within  $\min\{f + 2, t + 1\} - l$  rounds if  $l < f$  holds, within 2 rounds if  $l \geq f$  holds, and within 1 round if  $f = 0$  and  $l \geq t$  holds. On the other hand, the algorithm **ACC-ANF** terminates within  $\max\{3, \min\{f + 3, t + 1\} - l\}$  rounds if  $l < f$  holds. However, it requires no assumption on the number of faults, that is, the algorithm **ACC-ANF** can tolerate any number of crash faults. The comparison of our algorithm with previous algorithms is summarized in Table 1, where  $l$  is the legality level of input vectors, and  $d$  is the design parameter of each algorithm. The important points are that only our algorithms are adaptive to the conditions, and that the algorithm **FACC** achieves the best time complexity of all algorithms for the case  $t < n/2$ .

### 1.3 Related Work

Following the first result of the condition-based approach [17], several works have investigated this approach. For asynchronous uniform consensus, Mostefaoui et al. [21] show the tradeoff between efficiency of algorithms and degree of restriction to inputs, and explore several types of good conditions. Similar tradeoff in synchronous uniform consensus is studied in several previous works [17, 18, 25]. Our result also lies on this direction. From practical aspects, the efficiency of the condition-based approach is experimentally evaluated [15]. The problems other than the consensus are also investigated in several papers. Friedman et al. consider the condition-based  $k$ -set consensus problem [20]. Friedman et al. treat the problem of interactive consistency [10]. It also clarifies the relation between conditions and error-correcting codes, and studies the condition-based approach in the case of byzantine-failure models. Most recently, Mostefaoui et al. proposed

<sup>1</sup>The notion of the legal condition sequence and the legality level is similar to that of the *hierarchy* and the *degree* proposed in [21].

Table 1: The worst-case round complexity.

	$l \geq t$	$t > l \geq f$	$f > l$	Assumption
Zibin, 2003 [25]	$t + 2 - d$ (if $l \geq d$ ) $t + 1$ (otherwise)	$t + 2 - d$ (if $l \geq d$ ) $t + 1$ (otherwise)	$t + 2 - d$ (if $l \geq d$ ) $t + 1$ (otherwise)	$t < n/2$
Mostefaoui et al., 2003 [17]	1 if $f = 0$ 2 otherwise	$\min\{f + 2, t + 1\}$	$\min\{f + 2, t + 1\}$	$t < n$
Mostefaoui et al., 2004 [18]	2	$\min\{f + 2, t + 1 - d\}$ (if $l \geq d$ ) $\min\{f + 2, t + 1\}$ (otherwise)	$\min\{f + 2, t + 1 - d\}$ (if $l \geq d$ ) $\min\{f + 2, t + 1\}$ (otherwise)	$t < n$
this paper(FACC)	1 if $f = 0$ 2 otherwise	2	$\min\{f + 2, t + 1\} - l$	$t < n/2$
this paper(ACC-ANF)	1 if $f = 0$ 3 otherwise	3	$\max\{3, \min\{f + 3, t + 1\} - l\}$	$t < n$

$f$  : Actual number of faulty processes,  $t$  : Maximum number of faulty processes,  
 $l$  : Legality level of the input vector,  $d$  : Design parameter of each algorithm.

one advanced result about the condition-based  $k$ -set consensus [19]. This result investigates the solvability of condition-based  $k$ -set consensus on the system with some kind of failure detectors.

Besides the condition-based approach, such “input-sensitive” or “input-restriction” techniques are investigated in several works: To investigate the problem solvability, Taubenfeld and Moran consider the restriction of inputs in the several kinds of agreement problem [24]. Attiya et al. clarify the necessary and sufficient restriction of inputs such that the wait-free  $(n - 1)$ -set consensus problem is solvable [1]. While it is well-known that any run of asynchronous uniform consensus algorithms requires at least two communication steps, some algorithms can terminate within one communication step for some inputs. Brasileiro et al. present such input-sensitive approach [4]. Guerraoui and Raynal study the combination of the input-sensitive approach and other ones, such as unreliable failure detectors or randomization [12].

There are many works studying the (non-condition-based) consensus problem. Many papers have described consensus algorithms in synchronous systems subject to a certain class of faults [8, 22]. Raynal gives a brief survey of the synchronous consensus [23].

## 1.4 Roadmap

The paper is organized as follows: Section 2 provides the system model, the definition of the consensus problem, and the formalization of adaptive condition-based approach. In Section 3 and 4, we present our adaptive condition-based consensus algorithm FACC. This algorithm is presented in an incremental manner: In Section 3, we first show a basic adaptive condition-based algorithm BACC. The algorithm BACC includes the essence of condition adaptation. However, it has several rooms for improvement of time complexity. Thus, in Section 4, we present the algorithm FACC as an optimized version of BACC. The algorithm ACC-ANF is presented in Section 5. This algorithm is induced by modifying the algorithm FACC. In Section 6, we refer to open problems that is related to our result. Finally, we conclude this paper in Section 7.

## 2 Preliminaries

### 2.1 Distributed System

We consider a synchronous distributed system with round-based synchrony. The distributed system consists of  $n$  processes  $P = \{p_0, p_1, p_2, \dots, p_{n-1}\}$  that are completely connected, that is, any pair of processes can communicate with each other by directly exchanging messages. All channels are reliable: there is no creation, alteration, loss, or duplication of messages. The system is round-based, that is, its execution is a sequence of synchronized *rounds* identified by 1, 2, 3, etc. Each round  $r$  consists of three phases:

**Send phase** Each process  $p_i$  sends messages.

**Receive phase** Each process  $p_i$  receives all the messages sent to  $p_i$  at the send phase of round  $r$ .

**Local processing phase** Each process  $p_i$  executes local computation.

Processes can crash. If a process  $p_i$  crashes during round  $r$ , it makes no operation subsequently. The messages sent by  $p_i$  at round  $r$  may or may not be received. We say a process is *correct* if it never crashes, and say “a round  $r$  is correct” when no process crashes during round  $r$ . There are an upper bound  $t$  on the number of processes that can crash. We also denote the actual number of crash processes by  $f$  ( $\leq t$ ). Since, the algorithm **FACC** requires majority of processes are correct, we assume that  $t < n/2$  holds in Section 3 and 4. In contrast, the algorithm **ACC-ANF** requires no assumption, and thus we assume  $t < n$  in Section 5.

### 2.2 Uniform Consensus

In a consensus algorithm, each correct process initially proposes a value, and eventually chooses a decision value from the values proposed by processes so that all correct processes decide the same value. In this paper, we consider the uniform consensus problem, that is a stronger variant of the consensus problem. It disallows faulty processes to disagree on the decision value. More precisely, the uniform consensus is specified as follows:

*Termination* : Every correct process eventually decides.

*Uniform Agreement* : No two processes decide different values.

*Validity* : If a process decides a value  $v$ , then,  $v$  is a value proposed by a process.

In what follows, we often use “consensus” instead of “uniform consensus” for short. The set of values that can be proposed is denoted by  $\mathcal{V}$ . Moreover, we assume that  $\mathcal{V}$  is a finite ordered set.

### 2.3 Legality Level

#### 2.3.1 Notations

An *input vector* is a vector in  $\mathcal{V}^n$ , where the  $i$ -th entry represents  $p_i$ 's proposal value. We usually denote an input vector for an execution by  $I$ . We also define *view*  $J$  of  $I$  to be a vector in

$(\mathcal{V} \cup \{\perp\})^n$  obtained by replacing some entries in  $I$  by  $\perp$  ( $\perp$  is a default value such that  $\perp \notin \mathcal{V}$ ). Let  $\perp^n$  be the view such that all entries are  $\perp$ . For views  $J_1$  and  $J_2$ , we denote  $J_1 \leq J_2$  if  $\forall k : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$  holds. For two views  $J_1$  and  $J_2$  of an input vector, we define their union  $J = J_1 \cup J_2$  as follows:  $\forall k : J[k] = a \neq \perp \Leftrightarrow J_1[k] = a$  or  $J_2[k] = a$ . For a view  $J$  ( $\in (\mathcal{V} \cup \{\perp\})^n$ ) and a value  $a$  ( $\in \mathcal{V} \cup \{\perp\}$ ),  $\#_a(J)$  denotes the number of entries of value  $a$  in the vector  $J$ , that is  $\#_a(J) = |\{k \in \{0, 1, \dots, n-1\} | J[k] = a\}|$ . For a view  $J$  and a value  $a$ , we often describe  $a \in J$  if  $\#_a(J) \geq 1$ . Finally, for two vectors  $J_1$  and  $J_2$ , we denote the Hamming distance between  $J_1$  and  $J_2$  by  $\text{dist}(J_1, J_2)$ , that is  $\text{dist}(J_1, J_2) = |\{k \in \{0, 1, \dots, n-1\} | J_1[k] \neq J_2[k]\}|$ .

### 2.3.2 Conditions and Legality

A *condition* is formally defined as a subset of  $\mathcal{V}^n$ . First, as an important class of conditions, we introduce  $(d, h)$ -legal conditions <sup>2</sup>.

**Definition 1 (( $d, h$ )-legal conditions)** A condition  $C$  is  $(d, h)$ -legal (where  $h$  is a mapping  $h : C \mapsto \mathcal{V}$ ) if  $h$ ,  $d$ , and  $C$  satisfy the following properties:

1.  $\forall I \in C : \#_{h(I)}(I) > d$ ,
2.  $\forall I_1, I_2 \in C : h(I_1) \neq h(I_2) \Rightarrow \text{dist}(I_1, I_2) > d$ .

Intuitively, a  $(d, h)$ -legal condition is the set of input vectors  $I$  such that  $h(I)$  can be calculated even when at most  $d$  entries of  $I$  are lost. From the definition, as long as mapping  $h$  satisfies  $h(I) \in I$ ,  $\mathcal{V}^n$  can be  $(0, h)$ -legal, and  $(n, h)$ -legal conditions are only the empty set. Notice that  $(d, h)$ -legal conditions are not uniquely determined by  $d$  and  $h$  (for instance, for a  $(d, h)$ -legal condition, its subset is also a  $(d, h)$ -legal condition). In recent researches, it is shown that  $(d, h)$ -legal conditions help to reduce the worst-case execution time of synchronous consensus algorithms. To be more precise, for any  $(d, h)$ -legal condition, there exists a consensus algorithm that terminates (1) within  $\min\{t+1-d, f+2\}$  rounds for input vectors satisfying the condition, and (2) within  $\min\{f+2, t+1\}$  rounds otherwise [18]. In this sense, we can regard  $d$  as a characteristic value representing difficulties of input vectors in the  $(d, h)$ -legal condition. However, from the definition, a  $(d, h)$ -legal condition can include a  $(d+1, h)$ -legal condition. This implies that a  $(d, h)$ -legal condition can include easier input vectors. Therefore, to define actual difficulty of input vectors, we introduce *legality levels* of input vectors that are defined from a *legal condition sequence* as follows:

**Definition 2 (Legal condition sequence)** A sequence of conditions  $\mathcal{C} = \langle C_0, C_1, \dots, C_n \rangle$  is an  $h$ -legal condition sequence if the following properties are satisfied:

- $C_0 = \mathcal{V}^n$ ,  $C_n = \emptyset$ ,
- $\forall k$  ( $0 \leq k \leq n-1$ ):  $C_k$  is  $(k, h)$ -legal and  $C_{k+1} \subseteq C_k$ ,
- $\forall k$  ( $0 \leq k \leq n-1$ ): ( $\exists C' : C'$  is  $(k+1, h)$ -legal and  $C_{k+1} \subset C' \subseteq C_k$ ).

---

<sup>2</sup>The  $(d, h)$ -legal conditions is a subclass of  $d$ -legal conditions defined in [17] : the condition  $C$  is  $d$ -legal if there exists a mapping  $h$  such that  $C$  is  $(d, h)$ -legal. This difference does not restrict the class of condition applicable to our algorithm because our algorithm can be instantiated with any  $h$ .

**Definition 3 (Legality level)** For an  $h$ -legal condition sequence  $\mathcal{C}$ , the legality level of an input vector  $I$  is  $l$  if  $I \in C_l$  and  $I \notin C_{l+1}$  hold.

Since  $C_n$  is empty and  $C_0$  is the set of all possible input vectors, any legal condition sequence uniquely defines the legality level of each input vector. The legality level represents the actual difficulties of input vectors in the sense that we previously mentioned.

**Example** An example of a  $(d, h)$ -legal condition is  $C_d^{\max}$ :

$$C_d^{\max} = \{I \in \mathcal{V}^n \mid \#_a(I) > d, \text{ where } a \text{ is the maximum value in } I\}$$

The mapping  $h$  of  $C_d^{\max}$  is the function  $\max$ . The condition  $C_d^{\max}$  is a maximal  $(d, h)$ -legal condition, that is, there is no  $(d, h)$ -legal condition  $C$  such that  $C_d^{\max} \subset C$  [16]<sup>3</sup>. Therefore, for  $C_d^{\max}$ , we can define legal condition sequence  $\mathcal{S}^{\max} = \langle C_0^{\max}, C_1^{\max}, \dots, C_n^{\max} \rangle$ . As an example, we consider two input vectors,  $I_1 = \langle 0, 0, 1, 3, 3 \rangle$  and  $I_2 = \langle 0, 0, 2, 2, 2 \rangle$ . Both vectors are contained in  $C_1^{\max}$ . However, whereas  $I_2$  is contained in  $C_2^{\max}$ ,  $I_1$  is not. Therefore, for  $\mathcal{S}^{\max}$ , legality levels of vectors  $I_1$  and  $I_2$  are respectively 1 and 2.

The algorithm proposed in this paper is instantiated with a legal condition sequence, that is, the instantiated algorithm can utilize the legal condition sequence and the mapping  $h$ . In the following discussions, let the algorithm be instantiated with a legal condition sequence  $\mathcal{S} = \langle C_0, C_1, \dots, C_n \rangle$ , where each  $C_k$  is  $(k, h)$ -legal, and let  $l(I)$  be the legality level of an input vector  $I$  for  $\mathcal{S}$ .

### 3 Basic Adaptive Condition-Based Algorithm

In this section, we propose a condition-based consensus algorithm **BACC** that adapts to the legality level of input vectors. More precisely, the algorithm **BACC** terminates within 2 rounds if  $l(I) \geq f$  holds, and within  $f + 2 - l(I)$  rounds otherwise. In the following subsection, we first introduce the fundamental function **decode**, which is used as the subroutine of our algorithm. Then, we propose the algorithm **BACC**.

#### 3.1 Function decode

The function **decode**( $J$ ) is presented in Figure 1. The function **decode** has an argument  $J$  that is a view. Informally, the role of the function **decode**( $J$ ) is to obtain the value  $h(I)$  from view  $J$  of the input vector  $I$ . When a process invokes **decode**( $J$ ), it constructs all possible input vectors  $I'$  such that  $I' \geq J$  holds (line 4), and chooses the vector with the maximum legality level from them (line 5). For the chosen vector  $I'$ , the algorithm returns the value  $h(I')$ . For this function, we can show that the following lemmas hold. In the following lemmas, let  $\mathcal{E}(J)$  be the value stored in  $\mathcal{E}$  immediately after the line 4 of **decode**( $J$ ) is executed.

**Lemma 1** Letting  $\mathcal{E}(J)$  be nonempty,  $\text{dist}(I_1, I_2) \leq \#_{\perp}(J)$  holds for any  $I_1, I_2 \in \mathcal{E}(J)$ .

**Proof** Since  $J \leq I_1$  and  $J \leq I_2$ , this lemma clearly holds.  $\square$

<sup>3</sup>The original definition of maximality is stronger. In the original definition, the  $(d, h)$ -legal condition  $C$  is maximal if, for “any” mapping  $h'$ , there is no  $(d, h')$ -legal condition  $C'$  such that  $C \subset C'$ .

```

1: Function decode( $J$ ) :
2:   variable
3:      $\mathcal{E}$  : init  $\emptyset$ 

4:    $\mathcal{E} \leftarrow \{I \in V^n \mid h(I) \in J \text{ and } J \leq I\}$ 
5:    $I' \leftarrow \operatorname{argmax}_{I' \in \mathcal{E}} l(I')$ 
6:   return( $h(I')$ )

```

Figure 1: Function `decode`( $J$ )

**Lemma 2 (Decode Validity)** For any  $J \neq \perp^n$ , the resultant value of `decode`( $J$ ) is contained in  $J$ .

**Proof** If  $\mathcal{E}(J) \neq \emptyset$ , the resultant value  $h(I')$  is necessarily contained in  $J$  because any vector  $I'' \in \mathcal{E}(J)$  satisfies  $h(I'') \in J$ . Thus, we prove this lemma by showing  $\mathcal{E}(J) \neq \emptyset$  for any  $J \neq \perp^n$ . Suppose for contradiction that for a view  $J$ , the set  $\mathcal{E}(J)$  becomes empty. Then, we consider a vector  $I'' \geq J$  that is obtained by replacing  $\perp$  of  $J$  by a value in  $J$ . From the definition of mapping  $h$ ,  $h(I'')$  is a value in  $I''$ , that is, it is a value in  $J$ . This implies that  $I'' \in \mathcal{E}(J)$  holds. This is contradiction.  $\square$

The following is a key lemma of the function `decode`. It implies that if at most  $l(I)$  entries are missing in  $J$  (that is, at most  $l(I)$  processes crash), the function `decode` correctly calculates the value of  $h(I)$ .

**Lemma 3 (Decode Agreement)** Let  $I$  be an input vector, and  $J$  be a view such that  $J \leq I$  and  $\#_{\perp}(J) \leq l(I)$  hold. Then, `decode`( $J$ ) =  $h(I)$  holds.

**Proof** Let  $I'$  be the vector that is chosen from  $\mathcal{E}$  at line 5 ( that is, the resultant value is  $h(I')$ ). Since  $h(I)$  appears at least  $l(I)$  times in  $I$  and  $\#_{\perp}(J) \leq l(I)$  holds, we obtain  $h(I) \in J$ . Thus, the vector  $I$  is included in  $\mathcal{E}(J)$ . Then,  $\operatorname{dist}(I', I) \leq \#_{\perp}(J) \leq l(I)$  holds from Lemma 1. In addition, since the vector  $I'$  has the maximum legality level in  $\mathcal{E}(J)$ ,  $l(I) \leq l(I')$  also holds. This implies  $I' \in C_{l(I')} \subseteq C_{l(I)}$ . That is, both  $I'$  and  $I$  belong to a  $(l(I), h)$ -legal condition. From the definition of the  $(l(I), h)$ -legal condition, we obtain  $h(I') = h(I) = \text{decode}(J)$ .  $\square$

## 3.2 Algorithm BACC

Using the function `decode`, we construct a basic adaptive algorithm BACC. The algorithm BACC has two phases. The first phase consists of one round. The objective of this phase is to calculate an estimation of the decision value: at round one, each process  $p_i$  sends its proposal to all processes (including itself), and constructs a view  $J$  from all receiving messages. From the constructed view, each process estimates a decision value by using the function `decode`. That is, process  $p_i$  regards the resultant value of `decode`( $J$ ) as its current estimation. The second phase begins at round 2. In this phase, each process repeatedly exchanges its estimation with each other to reach the agreement. In each round  $r (\geq 2)$ , each process  $p_i$  sends its current estimation to all processes (including itself), and receives estimations from other processes. At the end of round  $r$ , it updates the estimation with one of the received estimations. The update rule is as follows: Basically,



```

Algorithm BACC( $v_i$ ) for  $h$ -legal condition sequence and  $t$  crashes ( $t < n/2$ )
Code for  $p_i$ :

1: variable:
2:    $S_i$  : init  $\perp^n$ 
3:    $s_i$  : init  $v_i$ 

4: for each round  $r = 1, 2, \dots, t + 2$  do :
5:   send  $s_i$  to all processes (including  $p_i$ )
6:   Let  $S_i[j]$  be the message received from  $p_j$ 
     (if no message is received from  $p_j$ ,  $S_i[j] = \perp$ )
7:   if  $r = 1$  then
8:      $s_i \leftarrow \text{decode}(S_i)$  /* Decoding */
9:   else
10:     $s_i \leftarrow \max(S_i)$  /* Updating the estimation */
11:    if  $\exists w \neq \perp : \#_w(S_i) > n/2$  then  $s_i \leftarrow w$  endif
12:    if  $\exists w \neq \perp : \#_w(S_i) + \#_\perp(S_i) = n$  then decide( $s_i$ ) and exit endif
13:  endif
14: endfor

```

Figure 2: Algorithm BACC: Basic Adaptive Condition-based Consensus

it updates the estimation with the maximum value of all received estimations. Exceptionally, if a process receives a same estimation  $w$  from more than  $n/2$  processes, it updates its current estimation with  $w$ . After the update, each process  $p_i$  determines whether it can decide or not. If all messages received by  $p_i$  contain a same estimation,  $p_i$  decides its current estimation (notice that the current estimation of  $p_i$  is same as one commonly contained in received messages). If  $p_i$  cannot decide, it proceeds to the next round. Each process continues the second phase until it decides or crashes.

Intuitively, the correctness of the algorithm BACC can be understood as follows: Since the estimation of each process  $p_i$  at round  $r$  is determined only by all estimations  $p_i$  receives at round  $r$ . Thus, if a round  $r$  is correct, each process receives a same set of messages, and thus each process estimates a common value. It follows that all non-crashed processes reach agreement at round  $r + 1$ . From this fact, we can easily show that each process decides by the round  $f + 2 - l(I)$  if more than  $l(I)$  processes crash at round one (because at least one round between 2 and  $f + 1 - l(I)$  is correct). On the other hand, if at most  $l(I)$  processes crash at round one, it is not guaranteed that a correct round exists by round  $f + 1 - l(I)$ . However, in this case, all processes have a view in which at most  $l(I)$  entries are lost at round 1. This implies that they can calculate the value  $h(I)$  by using the function **decode**. Then, all non-crashed processes have the estimation  $h(I)$  at round 1, and thus they reach agreement at round 2.

Figure 2 presents the code of the algorithm BACC for process  $p_i$ . The estimation of each process  $p_i$  is maintained in the variable  $s_i$ . The variable  $S_i$  denotes the set of received messages. It represents the view at round 1, and the collection of estimations at round 2 or later. The lines 10 and 11 correspond to the update of estimations, and the line 12 corresponds to the decision scheme. Notice that the decision is done after the update of estimations.

### 3.3 Correctness of BACC

In this subsection, we prove the correctness of BACC. For the proof, we define the following notations and terms:  $s_i^r$  and  $S_i^r$  respectively denote the values of  $s_i$  and  $S_i$  at the end of round  $r$ . Let  $P^r$  be the set of processes that neither crash by the end of round  $r$  nor terminate by the end of round  $r - 1$  (possibly terminate at the end of round  $r$ ), and  $P_c$  be the set of correct processes.

**Lemma 4 (Validity)** If a process decides a value  $w$ , then  $w$  is a value proposed by a process.

**Proof** If a process decides  $w$  at round  $r$ , then,  $w$  is the value of  $s_i^r$ , which is calculated by some process at round 1. That is, the value of  $s_i^r$  is a resultant value of the function **decode**. From Lemma 2, this lemma clearly holds.  $\square$

**Lemma 5** If  $s_i^r = w$  holds for any  $p_i \in P^r$ ,  $s_i^{r'} = w$  holds for any  $r' \geq r$  and  $p_i \in P^{r'}$ .

**Proof** We prove the lemma by induction on  $r'$ . (**Basis**) In the case of  $r' = r$ , the lemma clearly holds. (**Inductive step**) Suppose as induction hypothesis that  $s_i^{r'} = w$  holds for any  $p_i \in P^{r'}$ . The value of  $s_i^{r'+1}$  is determined at line 8, 10, or 11. In any case, from Lemma 2, a non- $\perp$  value in  $S_i^{r'+1}$  is assigned to  $s_i^{r'+1}$ . This implies that  $s_i^{r'+1} = w$  holds because each non- $\perp$  value in  $S_i^{r'+1}$  is a value of  $s_j^{r'}$  for some  $j$ , which is  $w$ .  $\square$

**Lemma 6** If  $s_i^r = w$  holds for any  $p_i \in P^r$  and a process  $p_j \in P^r$  decides, the decision value of  $p_j$  is  $w$ .

**Proof** Let  $r'(\geq r)$  be the round when  $p_j$  decides. Then, its decision value is  $s_i^{r'}$ . Since we obtain  $s_i^{r'} = w$  from Lemma 5,  $p_i$  decides  $w$ .  $\square$

**Lemma 7** If a round  $r$  ( $1 \leq r \leq t + 2$ ) is correct, then  $s_i^{r'} = s_j^{r'}$  holds for any  $r' \geq r$  and  $p_i, p_j \in P^{r'}$ .

**Proof** Since round  $r$  is correct, each process in  $P^r$  receives a same set of messages at round  $r$ . Thus, for any  $p_i, p_j \in P^r$ ,  $S_i^r = S_j^r$  holds. This implies  $s_i^r = s_j^r$  for any  $p_i, p_j \in P^r$  because the value of  $s_i^r$  is deterministically calculated from the values of  $S_i^r$ . Then, from Lemma 5,  $s_i^{r'} = s_j^{r'}$  holds for any  $r' \geq r$  and  $p_i, p_j \in P^{r'}$ .  $\square$

**Lemma 8** If at most  $l(I)$  processes crash at round 1, each process  $p_i$  decides at round 2 unless it crashes.

**Proof** Since at most  $l(I)$  processes crash at round 1,  $\#_{\perp}(S_k^1) \leq l(I)$  holds for any  $p_k \in P^1$ . From Lemma 3, we obtain  $s_k^1 = \text{decode}(S_k^1) = h(I)$ . Then, every process  $p_k$  in  $P^1$  sends the message  $h(I)$  at round 2, and thus,  $S_i^2$  contains only  $h(I)$  and  $\perp$ . This implies that  $p_i$  decides at round 2.  $\square$

**Lemma 9 (Termination)** (1) If  $l(I) \geq f$  holds, each process  $p_i$  decides a value at round 2 or earlier unless it crashes, and (2) if  $l(I) < f$  holds, each process  $p_i$  decides a value at round  $f + 2 - l(I)$  or earlier unless it crashes.

**Proof** The proof of (1) is straightforwardly obtained from Lemma 8. Thus, we have only to show the proof of (2). We consider the following two cases.

- **(Case1)** There exists a correct round  $r$  such that  $r \leq f + 1 - l(I)$ : From Lemma 7, the variable  $s_i^r$  has a common value (say  $s$ ) for any  $p_i \in P^r$ . Then, each process sends the same message  $s$  at round  $r + 1$ , and thus,  $S_i^{r+1}$  contains only  $s$  and  $\perp$ . It follows that each process  $p_i (\in P^{r+1})$  decides the value  $s$  at round  $r + 1 (\leq f + 2 - l(I))$  or earlier.
- **(Case2)** No round up to  $f + 1 - l(I)$  is correct: Since at least one process crashes in each round up to  $f + 1 - l(I)$ , at most  $l(I)$  processes crash at round 1. Then, from Lemma 8, each process  $p_i$  decides at round 2. Since we assume  $f > l(I)$ ,  $p_i$  decides at  $f + 2 - l(I)$  or earlier.

From the above, the lemma holds.  $\square$

**Lemma 10** Let  $p_i$  be the process that decides first. If  $p_i$  decides  $w$  at round  $r$ ,  $s_j^r = w$  holds for any  $p_j \in P^r$ .

**Proof** Let  $w$  be  $p_i$ 's decision value. Since  $p_i$  decides  $w$  at round  $r$ , all messages  $p_i$  receives at round  $r$  have  $w$ . In addition, since each process in  $P_c$  does not decide by the round  $r - 1$ , it sends the message  $w$  to all processes at round  $r$ . Thus, we obtain  $\#_w(S_j^r) > n/2$  for any  $p_j \in P^r$  because of  $|P_c| \geq n/2$ . Then, the value  $w$  is assigned to  $s_j^r$  by executing line 10. That is,  $s_j^r = w$  holds for any  $p_j \in P^r$ .  $\square$

**Lemma 11 (Uniform Agreement)** No two processes decide differently.

**Proof** Let  $p_i$  be the process that decides first,  $w$  be  $p_i$ 's decision value, and  $r$  be the round when  $p_i$  decides. We prove this lemma by showing that any process  $p_j$  has decision value  $w$  if it decides. From Lemma 10, we can show that  $w = s_i^r = s_j^r$  holds. Thus, from Lemma 6, we can conclude  $p_j$  decides  $w$ .  $\square$

From Lemma 4, 9, and 11, the following theorem holds.

**Theorem 1** For any input vector  $I$ , the algorithm BACC solves the uniform consensus (1) within  $f + 2 - l(I)$  rounds if  $l(I) < f$  holds, or (2) within 2 rounds otherwise.

## 4 Improved Algorithm

In this subsection, we introduce the algorithm FACC, which is a modified version of BACC. The algorithm FACC achieves better time complexity than BACC in some cases. It terminates within the same number of rounds (two if  $l(I) \geq f$  and  $f + 2 - l(I)$  if  $l(I) < f$ ) as BACC. In addition, it terminates within only one round if  $l(I) \geq t$  and  $f = 0$  hold, and terminates within  $t + 1 - l(I)$  rounds even if  $f = t$  holds. Therefore, the time complexity of FACC is  $\min\{f + 2, t + 1\} - l(I)$  rounds if  $l(I) < t$  holds, two rounds if  $l(I) \geq t$  holds, and one round if  $l(I) \geq t$  and  $f = 0$  hold.

## 4.1 Two Additional Decision Schemes

To improve the time complexity, the algorithm FACC uses two additional decision schemes, which is called *fast decision* and *slow decision*. These two schemes respectively correspond to one-round decision in the case of  $l(I) \geq t$  and  $f = 0$ , and  $(t + 1 - l(I))$ -round decision in the case of  $f = t$ . The idea of the fast decision scheme is borrowed from [17]. The basic idea of the slow decision scheme is proposed in a precedent result [18], and our scheme can be regarded as its adaptive version. The algorithm FACC is presented in Figure 3, which is developed by extending the code of BACC. In the figure, the lines with bold line numbers are additional or modified codes. The fast decision part appears at line 11, and the slow decision part appears at line 12, 14 and 18. In addition, transferred messages are modified to carry an extra information  $fn_i$ , which is used in the slow decision scheme. In the rest of this subsection, we describe the detailed behavior of these schemes.

### 4.1.1 Fast decision

At the first phase (that is, at round 1), if a process  $p_i$  gathers all proposals and recognizes that the legality level of the input vector is  $t$  or more, it does not have to execute the second phase. In this case, even though up to  $t$  processes crash, all other processes can calculate  $h(I)$  from their own views, and eventually decide a value  $h(I)$ . Thus, process  $p_i$  can immediately decide a value  $\text{decode}(J_i)$  ( $= h(I)$ ). This implies that the algorithm terminates within one round if  $f = 0$  and  $l(I) \geq t$ .

### 4.1.2 Slow decision

The aim of the slow decision scheme is to stop execution at round  $t + 2 - f'$ , where  $f'$  is the number of crash faults occurring at round one. It can be understood that this idea achieves the agreement within  $t + 1 - l(I)$  rounds by the following observation: If  $f'$  is  $l(I)$  or less, each process can calculate  $h(I)$  at round one, and thus they can reach agreement within two rounds. On the other hand, if  $f'$  is greater than  $l(I)$ , then  $(t + 2 - f')$ -round execution contains at least one correct round. This implies that all processes have a same estimation at the end of round  $t + 2 - f'$ , and thus they can reach agreement at round  $t + 2 - f'$  or earlier. Importantly, in the both cases, the length of the execution does not exceed  $t + 1 - l(I)$  rounds (recall that we consider only the case of  $l(I) < f$ ).

To achieve this aim, in our scheme, each process estimates  $f'$  by detecting crash faults occurring at round one. At round one, each process  $p_i$  counts the number of processes from which  $p_i$  receives no message, and stores it into the variable  $fn_i$ , which represents the estimation of  $f'$ . At round two or later, each process  $p_i$  broadcasts the current value of  $fn_i$  to all processes, and updates  $fn_i$  with the maximum value of all received  $fn_*$ . When current round  $r$  is larger than or equal to  $t + 2 - fn_i$ , process  $p_i$  decides its current estimation  $s_i$  and terminates. Each estimation  $fn_i$  can be slightly lower than the actual value of  $f'$ . However, this difference does not cause any problem. The details are explained in the correctness proof (the proof of Lemma 13).

## 4.2 Correctness of FACC

In this subsection, we prove the correctness of FACC. For the proof, we introduce the notation  $fn_i^r$  to denote the value of variable  $fn_i$  at the end of round  $r$ . We say that  $p_i$  decides by the *fast*

```

Algorithm FACC( $v_i$ ) for  $h$ -legal condition sequence and  $t$  crashes ( $t < n/2$ )
Code for  $p_i$ :

1: variable:
2:    $S_i$  : init  $\perp^n$ 
3:    $s_i$  : init  $v_i$ 
4:    $FN_i$  : init  $0^n$ 
5:    $fn_i$  : init 0

6: for each round  $r = 1, 2, \dots, t + 2$  do :
7:   send  $(s_i, fn_i)$  to all processes (including  $p_i$ )
8:   Let  $(S_i[j], FN_i[j])$  be the message received from  $p_j$ 
      (if no message is received from  $p_j$ ,  $S_i[j] = \perp$ )
9:   if  $r = 1$  then
10:     $s_i \leftarrow \text{decode}(S_i)$  /* Decoding */
11:     $fn_i \leftarrow \#_\perp(S_i)$  /* Counting #faults at round 1 */
12:    if  $\#_\perp(S_i) = 0$  and  $l(S_i) \geq t$  then decide( $s_i$ ) and exit endif /* Fast Decision */
13:  else
14:     $fn_i \leftarrow \max(FN_i)$ 
15:     $s_i \leftarrow \max(S_i)$  /* Updating the estimation */
16:    if  $\exists w \neq \perp$ :  $\#_w(S_i) > n/2$  then  $s_i \leftarrow w$  endif
17:    if  $\exists w \neq \perp$ :  $\#_w(S_i) + \#_\perp(S_i) = n$  then decide( $s_i$ ) and exit endif
18:    if  $r \geq t + 2 - fn_i$  then decide( $s_i$ ) and exit endif /* Slow Decision */
19:  endif
20: endfor

```

Figure 3: Algorithm FACC: Fast Adaptive Condition-based Consensus Algorithm

decision, the *slow decision* or the *regular decision* if  $p_i$  decides at line 12, 17, or 18 respectively. Lemmas 4, 6, 7, 8, and 9 also hold for FACC. Lemma 10 is slightly modified as follows (the proof is same as Lemma 10):

**Lemma 12** Let  $p_i$  be the process that decides first. If  $p_i$  decides  $w$  by the regular decision at round  $r$ ,  $s_i^r = w$  holds for any  $p_i \in P^r$ .

**Lemma 13 (Slow Termination)** If  $l(I) < f$  holds, each process  $p_i$  decides a value at round  $t + 1 - l(I)$  or earlier unless it crashes.

**Proof** Since  $l(I) < f \leq t$  holds,  $p_i$  does not decide at round 1. Let  $f_m$  be the value of  $fn_i^2$ . Then, we consider the following two cases.

- **(Case1)**  $l(I) < f_m$  holds: In this case, at round  $r = t + 1 - l(I)$ ,  $r \geq t + 2 - f_m \geq t + 2 - fn_i^r$  holds (notice that  $fn_i$  is non-decreasing). This implies that each process decides at round  $t + 1 - l(I)$  or earlier.
- **(Case2)**  $f_m \leq l(I)$  holds: Let  $P'$  be the set of processes from which  $p_i$  receives messages at round 2. Then, for any process  $p_j \in P'$ ,  $\#_\perp(S_j) = fn_j^1 \leq f_m (\leq l(I))$  holds from the following reason: if this does not hold, a process  $p_k$  in  $P'$  sends a message with the value of  $fn_k$  larger than  $f_m$  at round 2, and the value of  $fn_i^2$  becomes larger than  $f_m$ . Thus, from Lemma 3,  $\text{decode}(S_j^1) = h(I)$  holds for any process  $p_j \in P'$ . It follows that each process

$p_j \in P'$  sends the message  $(h(I), fn_j^1)$  at round 2. Then,  $S_i^2$  contains only  $h(I)$  and  $\perp$ . Therefore, we can conclude  $p_i$  decides at round  $2(\leq t + 1 - l(I))$ .

From the above, the lemma holds.  $\square$

**Lemma 14 (Fast Termination)** If  $l(I) \geq t$  and  $f = 0$  hold, each process  $p_i$  decides at round 1.

**Proof** Since  $f = 0$  holds, each process  $p_i$  receives messages from all processes. This implies that  $S_i^1 = I$  holds, and thus,  $p_i$  decides  $\text{decode}(I)$  at round 1 (line 12).  $\square$

**Lemma 15 (Uniform Agreement)** No two processes decide different values.

**Proof** Let  $p_i$  be the process that decides first,  $w$  be  $p_i$ 's decision, and  $r$  be the round when  $p_i$  decides. We show that any process  $p_j$  has decision value  $w$  if it decides. To show it, we prove  $s_k^r = w$  holds for any  $p_k \in P^r$ . If it holds, we can conclude that  $p_j$  decides  $w$  from Lemma 6. We consider the following cases:

- **(Case1)**  $p_i$  decides by the fast decision: Since  $p_i$  decides at round one,  $l(I) \geq t$  and  $w = h(I)$  clearly hold. Then, from Lemma 3,  $\text{decode}(S_k^1) = h(I) = w$  holds for any  $p_k \in P^1$  because  $\#_{\perp}(S_k^1) \leq t \leq l(I)$  holds. Thus, we obtain  $s_k^r = w$  for any  $p_k \in P^r$ .
- **(Case2)**  $p_i$  decides by the regular decision : In this case, the lemma clearly holds from Lemma 12.
- **(Case3)**  $p_i$  decides by the slow decision : Then,  $r \geq t + 2 - fn_i^r$  holds. Since at least  $fn_i^r$  processes crash at round one, there exists one correct round  $r'$  such that  $r' \leq r$ . Thus, from Lemma 7,  $s_k^r = s_i^r = w$  holds for any  $p_k \in P^k$ .

From the above, the lemma holds.  $\square$

From Lemmas 4, 9, 13, 14, and 15, the following theorem holds.

**Theorem 2** The algorithm FACC solves the uniform consensus (1) within one round if  $l(I) \geq t$  and no process crashes, (2) within two rounds if  $l(I) \geq f$ , and (3) within  $\min\{f + 2, t + 1\} - l(I)$  rounds otherwise.

## 5 Algorithm for Any Number of Faults

All previous sections required the assumption of  $t < n/2$ . In this section, we show that this assumption can be removed by sacrificing the time complexity. We present algorithm ACC-ANF, that is an adaptive condition-based consensus algorithm resilient to any number of faults (that is, for any  $t < n$ ). Compared with FACC, the algorithm ACC-ANF takes one extra round in some situations. The algorithm ACC-ANF also has the same three decision schemes as FACC; fast, regular, and slow decision. The fast decision scheme is completely same as FACC. That is, the algorithm ACC-ANF terminates within one round if  $f = 0$  and  $l(I) \geq t$  holds. The regular decision scheme needs one extra round to terminate. Formally, the algorithm terminates within  $f + 3 - l(I)$  rounds if  $l(I) < f$ , and within 3 rounds if  $l(I) \geq f$  holds. The slow decision scheme does not need extra rounds in almost all situations. Only if  $l(I) \geq t - 1$  holds, one extra round is imposed

to the slow decision scheme. That is, the slow decision scheme guarantees termination within  $\max\{3, t + 1 - l(I)\}$  rounds. From the above, the time complexity of the algorithm **ACC-ANF** is described as follows: it terminates within  $\max\{3, \min\{f + 3, t + 1\} - l(I)\}$  rounds if  $l(I) < f$ , within three rounds if  $l(I) \geq f$ , and within one round if  $l(I) \geq t$  holds and no process crashes.

## 5.1 Algorithm ACC-ANF

Before the presentation of the detail of **ACC-ANF**, we first states what role the assumption  $t < n/2$  plays in **FACC**. In simple terms, the role of this assumption is to guarantee the uniform agreement property. The algorithm **FACC** requires the assumption in the update of estimations (line 16 in Figure 3). That is, when a process decides  $v$ , the existence of more than  $n/2$  correct processes guarantees that other processes estimate the same value  $v$ . Thus, the key issue of **ACC-ANF** is to impose the same estimation with no use of the assumption  $t < n/2$  when a process decides (actually, the algorithm **ACC-ANF** consumes one extra round to handle this issue). To achieve it, Algorithm **ACC-ANF** uses a locking scheme: In the regular decision scheme of **ACC-ANF**, when a process  $p_i$  can decide  $v$  (that is, when all messages received by  $p_i$  contain a common estimation  $v$ ), it does not immediately decide, but it “locks” its estimation. The meaning of the lock is that the locking process has no other choice but to decide its current estimation. The locking process  $p_i$  imposes the locked estimation to all other processes. When a process  $p_i$  locks its estimation, the locking process  $p_i$  broadcasts a **LOCKED** message with its locked estimation  $v$  to all other processes at the next round, and then decides  $v$ . The process receiving the **LOCKED** message updates its estimation with the one attached to the **LOCKED** message, that is  $v$ . Then, if two **LOCKED** messages with different estimations are received, the receiver can choose either of these estimations (actually, we can show that such a case never occurs).

In this scheme, if a process  $p_i$  decides  $w$ , it is guaranteed that all processes have the same estimation  $w$  because they receive the **LOCKED** messages with  $w$  from  $p_i$ . Thus, without the assumption  $t < n/2$ , we can guarantee the uniform agreement. However, in the regular decision scheme, the algorithm **ACC-ANF** requires one extra round to broadcast the **LOCKED** message. Thus the time complexity of the algorithm **ACC-ANF** becomes  $f + 3 - l(I)$  if  $l(I) < f$ , and three if  $l(I) \geq f$ . In addition, this extra round prevents the algorithm from terminating within two rounds. Thus, except for the fast decision scheme, the algorithm needs at least three rounds to terminate. That is, in the slow decision scheme, the time complexity also becomes  $\max\{3, t + 1 - l(I)\}$ .

The algorithm **ACC-ANF** is presented in Figure 4. The lines with bold line numbers are additional or modified codes. The difference from **FACC** is to augment the locking scheme. Each process  $p_i$  maintains the variable  $l_i$ . If  $p_i$ 's estimation is not locked, the value  $\perp$  is stored in  $l_i$ . When  $p_i$  locks its estimation, it stores the flag “**LOCKED**” into  $l_i$  (lines 19). The variable  $l_i$  is broadcast at every round (line 8). When a process locks at round  $r$ , it decides at the next round  $r + 1$  (line 21).

## 5.2 Correctness Proof of ACC-ANF

In this subsection, we prove the correctness of **ACC-ANF**. For the proof, we introduce the notation  $l_i^r$  to denote the value of  $l_i$  at the end of round  $r$ . When  $l_i^{r-1} = \perp$ ,  $l_i^r \neq \perp$  and  $s_i^r = w \neq \perp$  hold, we say that  $p_i$  locks the estimation  $w$  at round  $r$ . Lemmas 4, 6, 7 and 14 also hold for **ACC-ANF**. Lemma 9 is slightly modified as follows (the proof is almost same as that for **BACC**):

**Algorithm ACC-ANF**( $v_i$ ) for  $h$ -legal condition sequence and  $t$  crashes ( $t < n$ )  
**Code for**  $p_i$ :

```

1: variable:
2:    $S_i, L_i$  : init  $\perp^n$ 
3:    $s_i$  : init  $v_i$ 
4:    $FN_i$  : init  $0^n$ 
5:    $fn_i$  : init 0
6:    $l_i$  : init  $\perp$ 

7: for each round  $r = 1, 2, \dots, t + 2$  do :
8:   send  $(s_i, fn_i, l_i)$  to all processes (including  $p_i$ )
9:   Let  $(S_i[j], FN_i[j], L_i[j])$  be the message received from  $p_j$ 
      (if no message is received from  $p_j$ ,  $S_i[j] = \perp$ ,  $FN_i[j] = \perp$ , and  $L_i[j] = \perp$ )
10:  if  $r = 1$  then
11:     $s_i \leftarrow \text{decode}(S_i)$  /* Decoding */
12:    if  $\#_{\perp}(S_i) = 0$  and  $l(S_i) \geq t$  then decide( $s_i$ ) and exit endif /* Fast Decision */
13:     $fn_i \leftarrow \#_{\perp}(S_i)$  /* Counting #faults at round 1 */
14:  else
15:    if  $l_i \neq \text{LOCKED}$  then
16:       $fn_i \leftarrow \max(FN_i)$ 
17:       $s_i \leftarrow \max(S_i)$  /* Updating the estimation */
18:      if  $\exists k : L_i[k] = \text{TRUE}$  then  $s_i \leftarrow S_i[k]$  endif /* Overwriting the estimation */
      (if there exist two or more values of  $k$  satisfying  $L_i[k] = \text{TRUE}$ , any of them can be chosen)
19:      if  $\exists w \neq \perp : \#_w(S_i) + \#_{\perp}(S_i) = n$  then  $l_i \leftarrow \text{LOCKED}$  endif /* Locking the estimation */
20:    else
21:      if  $l_i = \text{LOCKED}$  then decide( $s_i$ ) and exit endif /* Regular decision */
22:    endif
23:    if  $r \geq t + 2 - fn_i$  then decide( $s_i$ ) and exit endif /* Slow decision */
24:  endif
25: endfor

```

Figure 4: Algorithm ACC-ANF: Adaptive Condition-based Consensus Algorithm for Any Number of Faults

**Lemma 16** (1) If  $l(I) \geq f$  holds, each process  $p_i$  locks its estimation at round 2 or earlier unless it crashes, and (2) if  $l(I) < f$  holds, each process  $p_i$  locks its estimation at round  $f + 2 - l(I)$  or earlier unless it crashes.

From this lemma, we obtain the following corollary:

**Corollary 1 (Regular Termination)** (1) If  $l(I) \geq f$  holds, each process  $p_i$  decides at round 3 or earlier unless it crashes, and (2) if  $l(I) < f$  holds, each process  $p_i$  decides at round  $f + 3 - l(I)$  or earlier unless it crashes.

**Lemma 17 (Slow Termination)** If  $l(I) < f$  holds, each process  $p_i$  decides at round  $\max\{3, t + 1 - l(I)\}$  or earlier unless it crashes.

**Proof** Since  $l(I) < f \leq t$  holds,  $p_i$  does not decide at round 1. Let  $f_m$  be the value of  $fn_i^2$ . Then, we consider the following two cases.



- **(Case1)**  $l(I) < f_m$  holds: In this case, at round  $r = t + 1 - l(I)$ ,  $r \geq t + 2 - f_m \geq t + 2 - fn_i^r$  holds (notice that  $fn_i$  is non-decreasing). This implies that each process decides at round  $t + 1 - l(I)$  or earlier.
- **(Case2)**  $f_m \leq l(I)$  holds: let  $P'$  be the set of processes from which  $p_i$  receives messages at round 2. Then, for any process  $p_j \in P'$ ,  $\#_{\perp}(S_j^1) = fn_j^1 \leq f_m (\leq l(I))$  holds from the following reason: if this does not hold,  $p_j$  sends a message with the value of  $fn_j$  larger than  $f_m$  at round 2, and the value of  $fn_i^2$  becomes larger than  $f_m$ . Thus, from Lemma 3,  $\text{decode}(S_j^1) = h(I)$  holds for any process  $p_j \in P'$ . It follows that each process  $p_j \in P'$  sends the message  $(h(I), fn_j^1)$  at round 2. Then,  $S_i^2$  contains only  $h(I)$  and  $\perp$ . Therefore, we can conclude  $p_i$  locks its estimation at round 2 and decides at round 3.

From the above, the lemma holds.  $\square$

**Lemma 18** If two processes  $p_i$  and  $p_j$  ( $p_i, p_j \in P^r$ ) respectively lock  $w$  and  $w'$  at a same round  $r$ , then  $w' = w$  holds.

**Proof** Since both  $p_j$  and  $p_i$  do not crash at round  $r$ ,  $S_i^r[j] = S_j^r[j] \neq \perp$  and  $S_i^r[i] = S_i^r[j] \neq \perp$  hold. In addition, since both  $p_i$  and  $p_j$  respectively lock  $w$  and  $w'$  at round  $r$ ,  $S_i^r[i] = S_i^r[j] = w$  and  $S_j^r[i] = S_j^r[j] = w'$  also hold. Therefore, we obtain  $w = w'$ .  $\square$

**Lemma 19 (Uniform Agreement)** No two processes decide different values.

**Proof** Let  $p_i$  be the process that decides first,  $r$  be the round when  $p_i$  decides, and  $w$  be  $p_i$ 's decision. We show that any process  $p_j$  has decision value  $w$  if it decides. To show it, we prove  $s_k^r = w$  holds for any  $p_k \in P^r$ . If it holds, we can conclude that  $p_j$  decides  $w$  from Lemma 6. We consider the following cases:

- **(Case1)**  $p_i$  decides by the fast decision: Since  $p_i$  decides at round one,  $l(I) \geq t$  and  $w = h(I)$  clearly hold. Then, from Lemma 3,  $\text{decode}(S_k^1) = h(I) = w$  holds for any  $p_k \in P^1$  because  $\#_{\perp}(S_k^1) \leq t \leq l(I)$  holds. Thus, we obtain  $s_k^r = w$  for any  $p_k \in P^r$ .
- **(Case2)**  $p_i$  decides by the regular decision : Let  $P'$  be the set of processes that lock their estimations at round  $r - 1$ . For any process  $p_k \in P^r$ , at least one entry in  $L_k^r$  (i.e.  $L_k^r[i]$ ) has the value LOCKED. Thus, any process  $p_k \in P^r$  updates its estimation  $s_k$  by executing line 19 at round  $r$ . Since  $p_i$  locks  $w$  at round  $r - 1$ , from Lemma 18, each process  $p_j$  in  $P'$  locks  $w$ . This implies that  $S_k^r[j]$  has the value  $w$  for any  $p_j$  such that  $L_k^r[j] = \text{LOCKED}$  holds. That is,  $p_k$  updates  $s_k$  with  $w$ , and thus  $s_k^r = w$  holds for any  $p_k \in P^r$ .
- **(Case3)**  $p_i$  decides by the slow decision : Then,  $r \geq t + 2 - fn_i^r$  holds. Since at least  $fn_i^r$  processes crash at round one, there exists one correct round  $r'$  such that  $r' \leq r$ . Thus, from Lemma 7,  $s_j^r = s_i^{r'} = w$  holds.

From the above, the lemma holds.  $\square$

From Lemma 4, 17, 19, and Corollary 1, we obtain the following theorem.

**Theorem 3** For any  $t < n$ , algorithm ACC-ANF solves the uniform consensus (1) within one round if  $l(I) \geq t$  and no process crashes, (2) within three rounds if  $l(I) \geq f$ , and (3) within  $\max\{3, \min\{f + 3, t + 1\} - l(I)\}$  rounds otherwise.

## 6 Open Problems

In this section, we mention two open problems remained in this paper. One of the open problems is the design of the algorithm that tolerates any number of crash faults and achieves the same time complexity as **FACC**. Unfortunately, in this paper, we succeed neither to construct such algorithm, nor to present its impossibility. The key issue to resolve this open problem is the design of the  $(f + 2 - l(I))$ -round decision scheme. The weakness of **ACC-ANF** in the time complexity are caused by the only one fact that **ACC-ANF** requires one more round (that is, it needs  $f + 3 - l(I)$  rounds to terminate if  $l(I) < f$ , or three rounds if  $l(I) \geq f$ ) in the regular decision scheme. Conversely, if we can reduce the time complexity of the regular decision scheme by one round, the algorithm with the same time complexity as **FACC** is obtained. However, it is nontrivial task. It should be noted that the construction of such algorithm remains open even if we consider non-adaptive algorithms. More precisely, on the assumption of  $t < n$ , it remains open whether there exists or not a condition-based consensus algorithm for a  $d$ -legal condition  $C$  that terminates within  $f + 2 - d$  rounds if the input vector  $I \in C$  or within  $f + 2$  rounds otherwise.

Another open problem is to clarify the input vectors that enable one-round decision. The algorithm in this paper can terminate within one round if  $l(I) \geq t$  and  $f = 0$ . In the case of  $f = 0$ , it is also proved that  $l(I) \geq t$  is necessary to terminate within one round [17]. In this sense, our algorithm is optimal. However, in the case of  $f > 0$ , we does not have the algorithm terminating within one round. The key idea of one-round decision in the case of  $f = 0$  is that each process can recognize  $l(I) \geq t$  because it gathers the complete information about the input vector. In contrast, in the case of  $f > 0$ , each process cannot recognize what the actual value of  $l(I)$  is. It can be understood by the following observation: Let us consider the case that the legal condition sequence is  $\mathcal{S}^{max}$ , and the input vector  $l(I)$  is  $\langle 0, 1, 1, 1, 1 \rangle$ . Clearly, the value of  $l(I)$  is three. Then, if process  $p_4$  crashes and thus sends no message to process  $p_0$ ,  $p_0$  constructs the view  $\langle 0, 1, 1, 1, \perp \rangle$ . However, from this view,  $p_0$  cannot recognize that  $l(I) = 3$  because the input vector  $I$  may be  $\langle 0, 1, 1, 1, 2 \rangle$ . If so, the legality level  $l(I)$  is zero. This implies that there is a possibility that some other process decides 2. Thus, consequently,  $p_0$  cannot decide 1 at round 1 even if it has the view  $\langle 0, 1, 1, 1, \perp \rangle$ .

## 7 Concluding Remarks

This paper advanced the condition-based approach by newly introducing the concept of adaptation on the time complexity of condition-based algorithms. It classifies all possible input vectors into a hierarchical sequence of conditions (*legal condition sequence*) according to their difficulties called *legality level*. For such hierarchy, adaptive algorithms achieve time complexity depending on legality level of input vectors. We proposed two adaptive condition-based consensus algorithms. Two proposed algorithms respectively have distinct advantages: The first algorithm **FACC** requires the assumption that the majority of processes are correct (that is,  $t < n/2$ ), and terminates within  $\min\{f + 2, t + 1\} - l(I)$  rounds if  $l(I) < f$ , within two rounds if  $l(I) \geq f$ , and in one rounds if  $l(I) \geq t$  and  $f = 0$  hold. Compared with previous condition-based algorithms, this algorithm achieves the best time complexity. The second algorithm **ACC-ANF** can tolerate any number of crash faults (that is  $t < n$ ), and terminates within  $\max\{3, \min\{f + 3, t + 1\} - l(I)\}$  rounds if  $l(I) < f$ , within three rounds if  $l(I) \geq f$ , and in one rounds if  $l(I) \geq t$  and  $f = 0$  hold.

**Acknowledgment** This work is supported in part by a JSPS, Grant-in-Aid for Scientific Research ((B)(2)15300017), and “The 21st Century Center of Excellence Program” of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

## References

- [1] H. Attiya and Z. Avidor. Wait-free  $n$ -set consensus when inputs are restricted. In *Proc. of 17th International Conference on Distributed Computing (DISC)*, volume 2508 of *LNCS*, pages 326–338. Springer-Verlag, 2002.
- [2] H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
- [3] M. Ben-Or. Another advantage of free choice : Completely asynchronous agreement protocols (extended abstract). In *Proc. of the 2nd annual ACM symposium on Principles of distributed computing (PODC)*, pages 27–30, 1983.
- [4] F. V. Brasileiro, F. Greve, A. Mostéfaoui, and M. Raynal. Consensus in one communication step. In *Proc. of 6th International Conference on Parallel Computing Techn (PACT)*, volume 2127 of *LNCS*, pages 42–50. Springer-Verlag, 2001.
- [5] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
- [6] B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *Journal of Algorithms*, 51(1):15–37, Apr 2004.
- [7] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.
- [8] D. Dolev, R. Reischuk, and R. Strong. Early stopping in byzantine agreement. *Journal of ACM*, 37(4):720–741, 1990.
- [9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.
- [10] R. Friedman, A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Distributed agreement and its relation with error-correcting codes. In *Proc. of 17th International Conference on Distributed Computing (DISC)*, volume 2508 of *LNCS*, pages 63–87. Springer-Verlag, 2002.
- [11] R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Proc. of 9th International Workshop on Distributed Algorithms(WDAG)*, volume 972 of *LNCS*, Sep 1995.
- [12] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004.
- [13] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

- [14] M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13:124–149, 1991.
- [15] A. Mostéfaoui, E. Mourgaya, P. R. Parvédy, and M. Raynal. Evaluating the condition-based approach to solve consensus. In *Proc. of International Conference on Dependable Systems and Networks (DSN)*, volume 2848 of *LNCS*, pages 541–550. Springer-Verlag, 2003.
- [16] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954, 2003.
- [17] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Using conditions to expedite consensus in synchronous distributed systems. In *Proc. of 17th International Conference on Distributed Computing (DISC)*, volume 2848 of *LNCS*, pages 249–263, Oct 2003.
- [18] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. The synchronous condition-based consensus hierarchy. In *Proc. of 18th International Conference on Distributed Computing (DISC)*, volume 3274 of *LNCS*, pages 1–15. Springer-Verlag, Oct 2004.
- [19] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. The combined power of conditions and failure detectors to solve asynchronous set agreement. In *Proc. of the 24th annual ACM symposium on Principles of distributed computing (PODC)*, pages 179–188, 2005.
- [20] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based protocols for set agreement problems. In *Proc. of 17th International Conference on Distributed Computing (DISC)*, volume 2508 of *LNCS*, pages 48–62. Springer-Verlag, 2002.
- [21] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. *Distributed Computing*, 17(1):1–20, 2004.
- [22] K. J. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, SE-12(3):477–482, 1986.
- [23] M. Raynal. Consensus in synchronous systems: A concise guided tour. In *Proc. of Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 221–228, 2002.
- [24] G. Taubenfeld and S. Moran. Possibility and impossibility results in a shared memory environment. *Acta Informatica*, 33(1):1–20, 1996.
- [25] Y. Zibin. Condition-based consensus in synchronous systems. In *Proc. of 17th International Conference on Distributed Computing (DISC)*, volume 2848 of *LNCS*, pages 239–248, Oct 2003.

## Biography

**Taisuke Izumi** received the M.E. degrees in computer science from Osaka University in 2003. He is now a student of Graduate School of Information Science and Technology, Osaka University. His research interests include distributed algorithms. He is a student member of ACM and IEEE.

**Toshimitsu Masuzawa** received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Osaka University during 1987-1994, and was an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST) during 1994-2000. He is now a professor of Graduate School of Information Science and Technology, Osaka University. He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993-1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, EATCS and the Information Processing Society of Japan.