One-Step Consensus Solvability

Taisuke Izumi and Toshimitsu Masuzawa

Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, 560-8531, Japan
{t-izumi, masuzawa}@ist.osaka-u.ac.jp

Abstract. While any fault-tolerant asynchronous consensus algorithm requires two communication steps even in failure-free executions, it is known that we can construct an algorithm terminating in one step for some good inputs (e.g. all processes propose a same value). In this paper, we present the necessary and sufficient constraint for the set of inputs for which we can construct an asynchronous consensus algorithm terminating in one step. Our investigation is based on the notion of the condition-based approach: it introduces conditions on input vectors to specify subsets of all possible input vectors and condition-based algorithms can circumvent some impossibility if the actual input vector satisfy a particular condition. More interestingly, conditions treated in this paper are adaptive. That is, we consider hierarchical sequences of conditions whose k-th condition is the set of input vectors for which the consensus can be solved in one step if at most k processes crash. The necessary and sufficient constraint we propose in this paper is one for such condition sequences. In addition, we present an instance of the sufficient condition sequences. Compared with existing constraints for inputs this instance is more relaxed.

1 Introduction

The *consensus* problem is one of fundamental and important problems for designing fault-tolerant distributed systems. In the consensus problem, 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*, a stronger variant of the consensus, further requires that faulty processes are disallowed to disagree (Uniform Agreement). The (uniform) consensus problem has many practical applications, e.g., atomic broadcast [3, 10], shared object [1, 11], weak atomic commitment [8] and so on. While it is a very important task to build an efficient consensus primitive on the system because of such applications, it has no deterministic solution in asynchronous systems subject to only a single crash fault [5]. Thus, to circumvent this impossibility, several approaches, such as *eventual synchrony*, *unreliable failure detectors*, and so on, have been proposed. However, even using such approach, it is not an easy task to solve the consensus problem "efficiently". One of commonly used measurements to evaluate efficiency of algorithms is *communication steps*, one of which is an execution period where each pair of processes can concurrently exchange messages at most once. In asynchronous systems with some assumptions to solve the consensus problem, it is proved that any fault-tolerant consensus algorithm requires at least two communication steps for decisions even in the run where no crash fault occurs [13].

To circumvent this two-step lower bound, some papers investigate consensus algorithms that achieve one-step decision in some good input cases. The first result of such investigations is published by Brasileiro et al. [2]. On the assumption of the underlying non-one-step consensus primitive, this paper proposes a simple algorithm that correctly solves the consensus problem for any input and that especially achieves the one-step decision if all processes propose a same value. In other results [4][9], the one-step decision scheme is also considered in the context of efficient combination with other schemes such as randomization and failure detectors. However, these results leave an interesting and important question as follows: for what input can the consensus problem be solved in one step?

In this paper, we address this question based on the notion of the *condition-based approach*. The principle of the condition-based 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 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 conditions for which the uniform consensus can be solved in asynchronous systems subject to crash faults [14]. More precisely, this result presented a class of conditions, called *d-legal conditions*, and proved that the d-legal conditions are the class of necessary and sufficient conditions that make the (uniform) consensus solvable in asynchronous systems where at most d processes can crash. In previous results, the condition-based approach is used to overcome several impossibility results in distributed agreement problems [6, 12, 15–20]. We also use the notion of the condition-based approach to overcome the two-step lower bound of asynchronous consensus. In the same way as [2], this paper assumes the underlying non-one-step consensus primitive. On this assumption, the main objective of our study is to clarify the class of conditions such that we can construct the algorithm that terminates in one step for the inputs belonging to the condition and that even terminates (but not in one step) for any input out of the condition.

The contribution of this paper is to characterize such class of the necessary and sufficient conditions that make the uniform consensus terminate in one step. More interestingly, the condition we consider in this paper is *adaptive* in the sense of our previous result [12]. In the adaptive condition-based approach, a restriction for inputs is not represented by a single subset of all possible inputs, but represented by a hierarchical sequence of conditions called *condition sequence*. An adaptive condition-based algorithm is instantiated by a condition sequence, and guarantees some property according to the rank of the input vector in the hierarchy of the given condition sequence. For example, the first result for the adaptive condition-based approach [12] considers time complexity lower bound in synchronous consensus. In this result, all input vectors are classified into some hierarchical condition sequence whose k-th condition is the set of input vectors that reduce the worst-case time complexity of synchronous consensus by k, and we construct the algorithm achieving time reduction according to the rank of the actual input vector in the hierarchy. This paper considers the consensus algorithm instantiated by an *one-step condition sequence*, whose k-th condition is the set of input vectors for which the algorithm can terminate in one-step even if at most k processes crash. We present a property of condition sequences called *one-step legality*, and prove that a condition sequence can become the one-step condition sequence of some algorithm if and only if it is one-step legal. We introduce *root adjacency graphs*, which is an analysis tool for specifying the property of one-step legality. The notion of root adjacency graphs is based on the idea of graph representation of conditions proposed in [14]. An root adjacency graph is also a graph representation of a condition sequence, and the one-step legality property for a condition sequence is defined as the characterization of its root adjacency graph. Additionally, we also propose an instance of one-step legal condition sequences. Compared with existing constraints (i.e., all processes propose a same value), this instance is more relaxed.

The paper is organized as follows: In section 2, we introduce the system model, the definition of the consensus problem, and other necessary formalizations. Sections 3 and 4 provide the characterization theorem of one-step consensus solvability and its correctness proof. In Section 5, we present an example of one-step legal condition sequences. We conclude this paper in Section 6.

2 Preliminaries

2.1 Asynchronous Distributed Systems

A distributed system consists of n processes $\mathcal{P} = \{p_0, p_1, p_2, \cdots, p_{n-1}\}$, in which any pair of processes can communicate with each other by exchanging messages. All channels are reliable; neither message creation, alteration, nor loss occurs. The system is asynchronous in the sense that there is no bound on communication delay.

Each process can crash. If a process crashes, it prematurely stops its execution and makes no operation subsequently. Each process can crash at any timing. A process that does not crash (even in the future) is called a *correct* process. We assume that there is some upper bound t on the number of processes that can crash in the whole system. Every process knows the value of t a priori. The actual number of crash faults is denoted by f. The value of f is unknown to each process.

Formally, a process is modeled as a state machine. The communication is defined by two events, $\mathsf{Send}_i(m, p_k)$, and $\mathsf{Receive}_i(m, p_k)$. The event $\mathsf{Send}_i(m, p_k)$ is one that p_i sends message m to the process p_k. The event $\mathsf{Receive}_i(m, p_k)$ is one that p_i receives the message m from p_k. A process crash is also defined as an event. An event Crash_i means the crash of process p_i. We also assume that algorithms we consider in this paper are *deterministic*, i.e., the state after a transition is uniquely determined by the triggering event and the state immediately before the transition.

2.2 Uniform Consensus

A (uniform) consensus algorithm provides each process p_i with two events, $\mathsf{Propose}_i(v)$ and $\mathsf{Decide}_i(v)$, as the interface to the upper application layer. In a consensus algorithm, each correct process p_i initially proposes a value v by $\mathsf{Propose}_i(v)$, and eventually chooses a decision value v' by $\mathsf{Decide}_i(v')$. Then, the decision value must be chosen from the values proposed by processes so that all processes decide a same value. More precisely, the consensus problem is specified by the following three properties:

Termination **:** Every process p_i eventually invokes $\mathsf{Decide}_i(v)$ unless it crashes.
Uniform Agreement **:** If two events $\mathsf{Decide}_i(v_1)$ and $\mathsf{Decide}_j(v_2)$ are invoked, $v_1 = v_2$ holds.
Validity **:** If $\mathsf{Decide}_i(v)$ is invoked, then $\mathsf{Propose}_j(v)$ is invoked by some process p_j.

We define $\mathcal{V}$ as the set of all possible proposal values. Throughout this paper, we assume that $\mathcal{V}$ is a finite ordered set. An input to consensus algorithms is represented by a vector whose i-th entry is the value of p_i's proposal value. We call it an *input vector*.

2.3 Uniform Consensus Primitive

This paper investigates the inputs for which consensus algorithms can decide in one step. However, it is well-known that the consensus problem cannot be solved in asynchronous systems subject to only one crash fault. Thus, we need some assumption to guarantee correct termination for arbitrary inputs. In this paper, same as the previous result [2], we assume that a uniform consensus primitive is equipped to the system. This assumption can be regarded as an higher abstraction of other standard assumptions, such as *unreliable failure detector* or *eventual synchrony*, which are sufficient ones to solve the consensus problem. On this assumption, our aim is to provide an algorithm that decides in one step for good inputs and that solves consensus (but not in one step) for any input with support of the underlying uniform consensus primitive. In the following discussion, two events, $\mathsf{UC\_propose}(v)$ and $\mathsf{UC\_decide}(w)$, are provided by the underlying consensus, which mean the proposal of value v and the decision with value w respectively.

2.4 Configurations and Executions

A system configuration c is represented by all processes' states and the set of messages under transmission. An execution of a distributed system is an alternative sequence of configurations and events $E = c_0, e_0, c_1, e_1, c_2 \cdots$. In this paper, we deal with *admissible* executions, where occurrences of send, receive, propose and decide of the underlying consensus, and crash events satisfy those semantics.

2.5 Nonessential Assumptions

The asynchronous system model introduced in this section is the standard one as defined in [3]. In this subsection, for ease of presentation, we introduce some additional assumptions into the model. Notice that the introduced assumptions do not essentially differentiate our model from standard ones. Throughout this paper, we assume the followings:

– There exists a discrete global clock, and that each event has one time when it occurs. This global clock is a fictional device. That is, each process does not have access to the global clock (thus, it adds no additional power to the model).
– Local processing delay is negligible (i.e., any local computation is instantaneously processed).
– Each process p_i invokes $\mathsf{Propose}_i(v)$ at time zero unless it initially crashes.
– Any message has at least one time unit delay.

2.6 One-Step Decision of Consensus Problem

In this subsection, we introduce the definition of one-step decision in the consensus problem.

An *initial message* is one that is sent at time zero. Intuitively, an initial message sent by a process p_i is one whose sending event is triggered by p_i's activation of the consensus algorithm. We say that a message m is *over* at time u if the receiver of m has received m or crashed at u. Let $ot(E)$ be the minimum time when all initial messages are over in execution E. Then, the prefix of the execution E by time $ot(E)$ (including transitions occurring at time $ot(E)$) is called the *one-step prefix* of E, and is denoted by *pref*(E). We also define $E(\mathcal{A}, I)$ as the set of all admissible executions of a consensus algorithm $\mathcal{A}$ whose input vectors are I.

Using the above definitions, we define one-step decision as follows:

Definition 1 (One-step Decision) A consensus algorithm $\mathcal{A}$ decides in one step for an input vector I if for any execution $E \in E(\mathcal{A}, I)$ all processes decide or crash in *pref*(E).

3 Characterization of One-Step Consensus Solvability

3.1 Notations

For an input vector I, we define a *view* J of I to be a vector in $(\mathcal{V} \cup \{\perp\})^n$ obtained by replacing several 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, the containment relation $J_1 \leq J_2$ is defined as $\forall k(0 \leq k \leq n-1) : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$. We also describe $J_1 < J_2$ if $J_1 \leq J_2$ and $J_1 \neq J_2$ hold. 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 view J, that is, $\#_a(J) = |\{k \in \{0, 1, \cdots, n-1\} | J[k] = a\}|$. For a view J and a value a, we often describe $a \in J$ if there exists a value k such that $J[k] = a$. For an input vector $I \in \mathcal{V}^n$, For two views J_1 and J_2, let $\mathrm{dist}(J_1, J_2)$ be the Hamming distance between J_1 and J_2, that is $\mathrm{dist}(J_1, J_2) = |\{k \in \{0, 1, \cdots, n-1\} | J_1[k] \neq J_2[k]\}|$. A *condition* is a subset of all possible input vectors $\mathcal{V}^n$.

Let $[V^n]_k$ be the set of all possible views where $\perp$ appears at most k times.

3.2 One-Step Condition Sequence

The objective of this paper is not only to clarify the static conditions that enable the consensus problem to terminate in one step, but also to provide such conditions in an adaptive fashion: the content of a condition varies according to the number of actual faults. To handle such adaptiveness, we introduce *condition sequences*. Formally, a condition sequence S is a hierarchical sequence of $t+1$ conditions $(C_0, C_1, C_2, \cdots, C_t)$ satisfying $C_k \supseteq C_{k+1}$ for any $k(0 \leq k \leq t-1)$. Then, we define *one-step condition sequences* as follows.

Definition 2 (One-Step Condition Sequence) The *one-step condition sequence* of a consensus algorithm $\mathcal{A}$ is the condition sequence whose k-th condition $(0 \leq k \leq t)$ is the set of input vectors for which the algorithm $\mathcal{A}$ decides in one step when at most k processes crash.

The one-step condition sequence of an algorithm $\mathcal{A}$ is denoted by $Sol^{\mathcal{A}}$.

3.3 Characterization Theorem

This subsection presents the characterization theorem for one-step consensus solvability. The key idea of the characterization theorem derives from the notion of *legality* in [14]: we consider a graph representation of condition sequences, and the characterization is given as a property of such graphs. To provide the theorem, we first introduce *root adjacency graphs* and their legality. Root adjacency graphs are a variant of the graph representation of legal conditions proposed in [14] so that it can handle the one-step solvability and condition sequences.

Definition 3 (Decidable/Undecidable views) For a condition sequence $S = (C_0, C_1, \cdots, C_t)$, the *decidable views* $DV(S)$ and *undecidable views* $UV(S)$ are respectively the set of views defined as follows:

$$DV(S) = \bigcup_{k=1}^{t} \{J | dist(J, I) \leq k, I \in C_k\}$$

$$UV(S) = \bigcup_{J \in DV(S)} \{J' | J' \in [V^n]_t, J' \leq J\}$$

Notice that $DV(S) \subseteq UV(S)$ holds.

Definition 4 (Root Adjacency Graphs) Given a condition sequence S, its root adjacency graph $RAG(S)$ is the graph such that

- The vertex set consists of all views in $UV(S)$.
- The two views J_1 and J_2 are connected if $J_1 \leq J_2$ holds and J_2 belongs to $DV(S)$.

The legality of root adjacency graphs is defined as follows:

Definition 5 (Legality) A root adjacency graph G is *legal* if, for each connected component Com of G, at least one common value appears at all views belonging to Com.

We say a condition sequence S is *one-step legal* if $RAG(S)$ is legal. Using the above definitions, we state the characterization theorem.

Theorem 1 (One-step Consensus Solvability Theorem) There exists a consensus algorithm whose one-step condition sequence is S if and only if S is one-step legal.

The intuitive meaning of root adjacency graphs is explained as follows: In general, the information that a process can gathers in one-step prefixes can be represented by a view because the information each process can send in one-step prefixes is only its proposal. In this sense, the decidable views can be interpreted as the set of views J such that if a process gathers the information corresponding J in one step , it can immediately decides (i.e., one-step decision). The undecidable views can also be interpreted as the set of views J such that if a process gathers the information corresponding J in one step, it must consider the possibility that other processes may decide in one step (but it does not have to decide immediately). Then, a root adjacency graph can be regarded as one obtained by connecting two views J_1 and J_2 such that if two processes gathers J_1 and J_2 respectively, they must reach a same decision. Thus, the sentence "root adjacency graphs is legal" implies that there exists at least one possible decision value.

For any view $J \in DV(S)$, there exists an input vector I satisfying $\mathsf{dist}(J, I) \leq k$ and $I \in C_k$ for some k. In such vectors, we call one that maximizes k the *master vector* of J (if two or more vectors maximize k, one chosen by some (arbitrary) deterministic rule is the master vector of J). Then, we also call the value of k *legality level* of the master vector. For any view $J \in DV(S)$ and its master vector I with legality level k, there exists a view J' satisfying $J' \leq J$ and $J \leq I$. The view J' minimizing $\#_\perp(J')$ is called the root view of J, and denoted by $Rv(J)$. Notice that $Rv(J) \in DV(S)$ and $\#_\perp(J') \leq k$ necessarily hold because of $I \in C_K$ and $\mathsf{dist}(J, I) \leq k$.

4 Proof of the Characterization Theorem

4.1 Proof of Sufficiency

This subsection presents the sufficiency proof of the theorem, i.e., we propose a generic one-step consensus algorithm for any one-step legal condition sequence S and prove its correctness.

Figure 1 presents the pseudo-code description of a generic consensus algorithm OneStep that is instantiated by any one-step legal condition sequence S. In the description, we use the function h, that is the mapping from a view in $UV(S)$ to a value in $\mathcal{V}$. The mapped value $h(J)$ for a view J is one that appears in common at any view in the connected component to which J belongs (such value necessarily exists from the fact that S is one-step legal). If two or more values appear in common, the largest one is chosen. The algorithmic principle of OneStep is as follows: first, each process p_i exchanges its proposal with each other, and constructs a view J_i. The view J_i is maintained incrementally. That is, it is updated on each reception of a message. When at least $n - t$ messages are received by p_i, process p_i tests whether J_i belongs to the undecidable views $UV(S)$ or not. If it belongs to $UV(S)$, process p_i activates the underlying consensus with proposal $h(J_i)$. Otherwise, p_i activates the underlying consensus with proposal $J_i[i]$. In addition, when the view J_i is updated, each process p_i tests whether its view J_i belongs to the decidable views $DV(S)$ or not. If J_i belongs to $DV(S)$, process p_i immediately decides $h(J_i)$, that is, it decides in one step. When the underlying consensus reaches decision, each process simply borrows the decision of the underlying consensus unless it has already decided in one step. Intuitively, the correctness of the algorithm OneStep relies on two facts: one is that if two processes p_i and p_j decide in one-step, the views J_i and J_j at the time of their decisions are necessarily connected in $RAG(S)$, and another one is that if a process p_i decides v in one-step, each process p_j activates the underlying consensus with the proposal value v. From the former fact, we can show that two processes p_i and p_j, both of which decide in one step, have a same decision. The latter fact implies that if a process p_i decides v in one step, any other process p_j (that may not decide in one step) propose v. The detailed explanation of correctness is given in the following proof.

Correctness We prove the correctness of the algorithm Onestep. In the following proofs, let vector J_i^{pro} and J_i^{dec} be the the value of J_i at the time when p_i execute the lines 13 and 10 respectively (Notice that both values are uniquely defined because lines 13 and 10 are respectively executed at most once). If p_i does not execute line 13 (10), J_i^{pro} (J_i^{dec}) is undefined.

Lemma 1 (Termination) Each process p_i eventually decides unless it crashes.

Proof Since at most t processes can crash, each process p_i receives at least $n-t$ messages. Then, p_i necessarily activates the underlying consensus, and thus the

```
Algorithm OneStep for one-step legal condition sequence S
Code for p_i:

1:  variable:
2:     proposed_i, decided_i : FALSE
3:     J_i : init ⊥^n

4:  Upon Propose_i(v_i) do:
5:     J_i[i] ← v_i
6:     Send v_i to all processes (excluding p_i);

7:  Upon Receive_i(v) from p_j do:
8:     J_i[j] ← v
9:     if J_i ∈ DV(S) and decided_i ≠ TRUE then
10:       decided_i ← TRUE ; Decide_i(h(J_i))
11:    endif
12:    if #_⊥(J_i) ≤ t and proposed_i = FALSE then
13:       if J_i ∈ UV(S) then UC_propose_i(h(J_i))    /* Starting the Underlying Consensus */
14:       else UC_propose(J_i[i]) endif
15:       proposed_i ← TRUE
16:    endif

17: Upon UC_decide_i(v) do:                           /* Decision of the Underlying Consensus */
18:    if decided_i ≠ TRUE then
19:       decided_i ← TRUE ; Decide_i(v)
20:    endif
```

Fig. 1. Algorithm Onestep: An One-Step Consensus Algorithm for a One-Step Legal Condition Sequence S

decision of underlying consensus eventually occurs on p_i unless p_i crashes. This implies that p_i eventually decides unless it crashes. $\square$

Lemma 2 (Uniform Agreement) No two processes decide differently.

Proof Let p_i and p_j be the processes that decide, and v_i and v_j be the decision values of p_i and p_j respectively. The input vector is denoted by I. Then, we prove $v_i = v_j$. We consider the following three cases.

- (**Case1**) When both p_i and p_j decide at line 10: For short, let $g = \#_\perp(J_i^{dec})$. Both J_i^{dec} and J_j^{dec} appear in $DV(S)$. Let I' be the master vector of J_i^{dec}, and k be its legality level. Then, for any vector I'' that is obtained from J_i^{dec} by replacing $\perp$ by any value, $\mathsf{dist}(I', I'') \leq k$ holds. Thus, letting I be the input vector, $\mathsf{dist}(I', I) \leq k$ holds, and thus we obtain $I \in DV(S)$. In addition, since $J_i^{dec} \leq I$, and $J_j^{dec} \leq I$ holds, I and J_i^{dec}, and I and J_j^{dec} are respectively adjacent to each other in $RAG(S)$. This implies that $v_i = h(J_i^{dec}) = h(J_j^{dec}) = v_j$ holds.
- (**Case2**) When p_i and p_j respectively decide at lines 10 and 19: Since p_j's decision is borrowed from the decision of the underlying consensus primitive, it is a value proposed by some process at line 13 or 14. Thus, it is sufficient to show that every process p_k proposes v_i at line 13 or 14 unless it crashes. Let v_k be p_k's proposal. J_i^{dec} appears in $DV(S)$. Then, by the same argument

as Case 1, we can show $I \in DV(S)$. Since $J_k^{pro} \leq I$ and $J_i^{dec} \leq I$ holds, we can conclude $J_k^{pro} \in UV(S)$, that is, p_k propose the value $h(J_k^{pro})$ at line 13. Since J_k^{pro} and J_i^{dec} is connected in $RAG(S)$, $v_k = h(J_k^{pro}) = h(J_i^{pro}) = v_i$ holds. It follows that every process p_k proposes v_i.

– (**Case3**) When both p_i and p_j decide at line 19: Then, from the uniform agreement property of the underlying consensus, $v_i = v_j$ clearly holds.

Consequently, the lemma holds. □

Lemma 3 (Validity) If a process decides a value v, then, v is a value proposed by a process.

Proof If a process p_i decides at line 15, its decision value is $h(J_i^{dec})$, which is a value in J_i^{dec}. On the other hand, if a process p_i decides at line 19, then, its decision value is a proposal of the underlying consensus. That is, the decision value is $h(J_k^{pro})$ or $J_k^{pro}[k]$ for some p_k, which is also a value in J_k^{pro}. In both cases, the decision value is one of proposals, and thus the validity holds. □

Lemma 4 (One-Step decision) The algorithm $\mathsf{OneStep}$ decides in one step for any input vector I belonging to $S_k (k \geq f)$ if at most k processes crash.

Proof Since each process p_i receives the messages from all correct processes unless it crashes, $\#_\perp(J_i) \leq k$ holds eventually. Then, J_i is included in $[S_k]_k$. This implies that p_i decides in one step. □

From Lemma 1, 2, 3, and 4, we can show the following lemma that implies the sufficiency of the characterization theorem.

Lemma 5 For any one-step legal condition sequence S, there exists one-step consensus algorithm whose one-step condition sequence is S.

4.2 Proof of Necessity

This subsection presents the necessity proof of the characterization theorem. In the proof we consider a subclass of admissible executions called *P-block synchronous executions*, which are defined as follows:

Definition 6 (P-Block Synchronous Executions) For a set of processes P, P-block synchronous executions are defined as ones satisfying the following properties:

1. No $\mathsf{UC\_decide}_i(v')$ occurs at time zero or one.
2. All message transferred between two processes in P have one time unit delay, and others have two time unit delay.

For a view, J, let $P(J)$ be a set of processes p_i such that $J[i] \neq \perp$ holds. For a consensus algorithm $\mathcal{A}$, a view J, and a set of processe P, let $E_{Sync}(\mathcal{A}, J, P)$ be the set of all possible P-block synchronous executions where process $p_i \in P(J)$ proposes $J[i]$ and never crashes, and $p_i \notin P(J)$ initially crashes. Here, we define *representative executions* of a view J as follows:

Definition 7 (Representative Executions) For a consensus algorithm $\mathcal{A}$ and a view J, its representative executions $E_{Rep}(\mathcal{A}, J)$ is a set of executions defined as follows:

$$E_{Rep}(\mathcal{A}, J) = \begin{cases} E_{Sync}(\mathcal{A}, J, \mathcal{P}) & \text{if } J \notin DV(Sol^{\mathcal{A}}) \\ E_{Sync}(\mathcal{A}, J, P(Rv(J))) & \text{if } J \in DV(Sol^{\mathcal{A}}) \end{cases}$$

For a consensus algorithm $\mathcal{A}$ and a view J, let $val(\mathcal{A}, J)$ be the set of all decision values that appear at executions in $E_{Rep}(\mathcal{A}, J)$.

Using the above notations, we prove the following lemma that implies the necessity of the characterization theorem (for lack of space, we only give a part of all proofs).

Lemma 6 Let $\mathcal{A}$ be an consensus algorithm, and J be a view in $DV(Sol^{\mathcal{A}})$. Then, in any execution $E \in E_{Rep}(\mathcal{A}, J)$, each process in $P(Rv(J))$ decides in one step.

Proof Let I be the master vector of J, and k be its legality level. From the definition of root views, $Rv(J) \leq I$ holds. Since $\#_{\perp}(Rv(J)) \leq k$ holds, in any execution $E \in E_{Rep}(\mathcal{A}, J)$, at most k processes crash. Thus, any execution $E \in E_{Rep}(\mathcal{A}, J)$ can be regarded as one where input vector is $I \in C_k$ and the number of crash processes is at most k. This implies that each process achieves one-step decision in E.

Lemma 7 Let $\mathcal{A}$ be an consensus algorithm. Then, $RAG(Sol^{\mathcal{A}})$ is one-step legal.

Proof Clearly for any view J, all values in $val(J)$ must be appeared in J from the validity property of the consensus problem. Thus, we prove this lemma by showing $val(J_1) = val(J_2)$ for any two different views J_1 and J_2 that are adjacent to each other in $RAG(Sol^{\mathcal{A}})$. Then, a value in $val(J)$ appears in any view of the connected component to which J belongs, i.e. each connected component has a common value.

Suppose for contradiction that $val(J_1) \neq val(J_2)$ holds for two different views J_1 and J_2 that are adjacent to each other. Without loss of generality, we assume $J_2 < J_1$. Then, from the definition of the $RAG(Sol^{\mathcal{A}})$, J_1 belongs to $DV(Sol^{\mathcal{A}})$. Since we assume $val(J_1) \neq val(J_2)$, there exist two synchronous executions $E_1 \in E_{Rep}(\mathcal{A}, J_1)$ and $E_2 \in E_{Rep}(\mathcal{A}, J_2)$ where processes reach different decisions v_1 and v_2 respectively. In execution E_2, all correct processes eventually reach to the decision. Let Δ be the time when all correct processes decides in E_2. From the

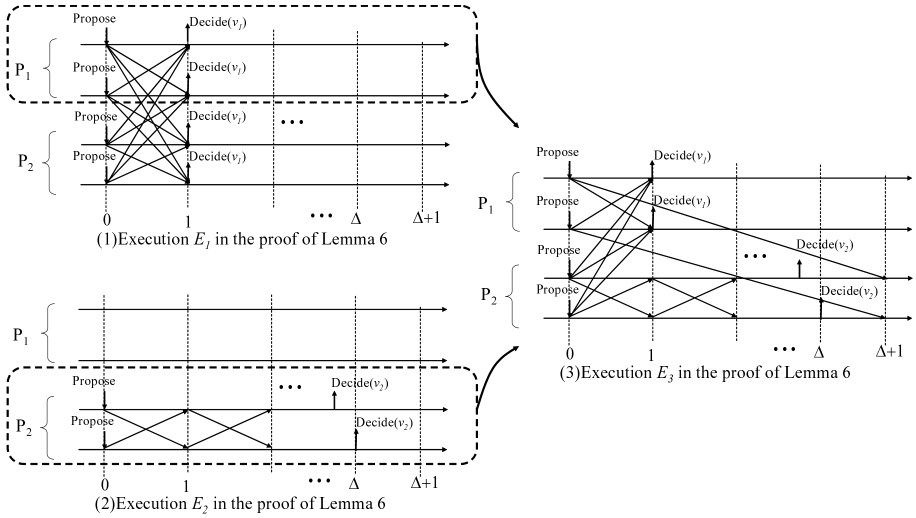

Fig. 2. Executions E_1, E_2 and E_3

fact of $J_2 < J_1$ and $J_1 \leq Rv(J_1)$, there exists at least one process in $P(Rv(J_1))$ that does not crash in E_1 but crashes in E_2. Let P_1 be the set of such processes. We also define P_2 as $\mathcal{P} - P_1$. Then, we consider the execution E_3 obtained by modifying the execution E_2 as follows:

- The behavior of each process in P_2 by time Δ is identical to E_2.
- Each process p_i in P_1 proposes a value $J_1[i]$.
- All messages transferred from a process in P_1 to one in P_2 have $\Delta + 1$ time unit delay, and all other messages have exactly one time unit delay.

The construction of the execution E_3 is illustrated in Figure 2. Since each process in P_1 does not affect ones in P_2 by time $Delta + 1$, the execution E_3 is possible admissible execution of the algorithm $\mathcal{A}$. In execution E_3, each non-faulty process in P_2 decides v_2 at Δ or earlier because it cannot distinguish the execution E_3 from E_2 by time $\Delta + 1$. In both E_1 and E_3, each process in P_1 receives a same set of initial messages at time one because initial messages sent by a process p_i is uniquely determined by p_i's proposal. This implies that each process in P_1 cannot distinguish E_3 from E_1 by time two. Thus, by Lemma 6, in E_3 each process in P_1 decides v_1 at time one. However, since we assume $v_1 \neq v_2$, the execution E_3 has two different decision values (processes in P_1 decides v, and others decides v_2). It contradicts the uniform agreement property of the consensus. $\square$

5 An Example of One-Step Legal Condition Sequence

In this section, we propose an example of one-step legal condition sequences. First, we introduce a condition that are basis of the example.

Definition 8 (Frequency-Based Condition C_d^{freq}) Let $1\mathrm{st}(J)$ be the non-$\perp$ value that appears most often in view J (if two more values appears most often, the largest one is chosen), and $\hat{J}$ be the vector obtained from J by replacing $1\mathrm{st}(J)$ by $\perp$. Letting $2\mathrm{nd}(J) = 1\mathrm{st}(\hat{J})$, the frequency-based condition C_d^{freq} is defined as follows:

$$C_d^{\mathrm{freq}} = \{I \in \mathcal{V}^n | \#_{1\mathrm{st}(I)}(I) - \#_{2\mathrm{nd}(I)}(I) > d\}$$

It is known that C_d^{freq} belongs to *d-legal* conditions, which is the class of conditions that are necessary and sufficient to solve the consensus problem in failure-prone asynchronous systems where at most d processes can crash.

Using this condition, we can construct a one-step condition sequence.

Theorem 2 Letting $3t < n$, a condition sequence, $Sa^{\mathrm{freq}} = (C_t^{\mathrm{freq}}, C_{t+2}^{\mathrm{freq}}, \cdots, C_{t+2k}^{\mathrm{freq}}, \cdots, C_{3t}^{\mathrm{freq}})$ is one-step legal.

Proof We prove the one-step legality of Sa^{freq} by showing that $1\mathrm{st}(J_1) = 1\mathrm{st}(J_2)$ holds for any two views J_1 and J_2 that are adjacent to each other in $RAG(Sa^{\mathrm{freq}})$. Suppose $1\mathrm{st}(J_1) \neq 1\mathrm{st}(J_2)$ for contradiction. Since either J_1 or J_2 belongs to $DV(Sa^{\mathrm{freq}})$, we assume $J_1 \in DV(Sa^{\mathrm{freq}})$ without loss of generality. Let I_1 be the master vector of J_1, and k be its legality level. From the definition of C_{t+2k}^{freq}, $\#_{1\mathrm{st}(I_1)}(I_1) - \#_v(I_1) > t + 2k$ holds for any value $v \neq 1\mathrm{st}(I_1)$. Then, since $dist(J_1, I_1) \leq k$ holds, $\#_{1\mathrm{st}(I_1)}(J_1) - \#_v(J_1) > t$ also holds for any value v. This implies $1\mathrm{st}(I_1) = 1\mathrm{st}(J_1)$, and thus we obtain $\#_{1\mathrm{st}(J_1)}(J_1) - \#_v(J_1) > t$ for any v. It follows that $\#_{1\mathrm{st}(J_1)}(J_2) - \#_{1\mathrm{st}(J_2)}(I_1) > 0$ holds because of $J_1 \geq J_2$ and $\#_\perp(J_2) \leq t$. However, it contradicts to the fact that $1\mathrm{st}(J_2)$ is the most often value in J_2. Thus, $1\mathrm{st}(J_2) = 1\mathrm{st}(J_1)$ holds and the lemma is proved. $\square$

Notice that the assumption $3t < n$ is necessary is to achieve one-step decision [2]. That is, if $3t \geq n$, no condition sequence is one-step legal.

Compared with existing constraints (e.g., one that all processes propose a same value), the adaptive condition sequence Sa^{freq} is more relaxed. Thus, the algorithm OneStep instantiated by Sa^{freq} is a strict improvement of existing one-step consensus algorithms.

6 Concluding Remarks

This paper investigated the one-step solvability of consensus problem. While any consensus algorithm require at least two steps even in failure-free executions, we can construct an algorithm that terminates in one step for several good inputs.

In this paper, we proposed the necessary and sufficient condition sequences for which we can construct one-step consensus algorithms. We also presented an instance of sufficient condition sequences. Compared with existing constraints for inputs (e.g., all processes propose a same value), this instance is more relaxed.

Acknowledgment This work is supported in part by MEXT: "The 21st Century Center of Excellence Program", JSPS: Grant-in-Aid for Scientific Research ((B)15300017), MEXT: Grant-in-Aid for Scientific Research on Priority Areas (16092215) and MIC: Strategic Information and Communications R&D Promotion Programme (SCOPE).

References

1. H. Attiya and J. L. Welch. Sequential consistency versus linearizability. *ACM Transactions on Computer Systems*, 12(2):91–122, 1994.
2. 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.
3. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.
4. P. Dutta and R. Guerraoui. Fast indulgent consensus with zero degradation. In *Proc. of the 4th European Dependable Computing Conference on Dependable Computing(EDCC)*, volume 2485 of *LNCS*, pages 191–208. Springer-Verlag, 2002.
5. 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.
6. 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 Ditributed Computing (DISC)*, volume 2508 of *LNCS*, pages 63–87. Springer-Verlag, 2002.
7. E. Gafni. Round-by-round faults detectors (extended abstract): unifying synchrony and asynchrony. In *Proceedings of the 17th annual ACM symposium on Principles of distributed computing*, pages 143–152, 1998.
8. 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.
9. R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Trans. Computers*, 53(4):453–466, 2004.
10. 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.
11. M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13:124–149, 1991.
12. T. Izumi and T. Masuzawa. Synchronous condition-based consensus adapting to input-vector legality. In *Proc. of 18th International Conference on Distributed Computing(DISC)*, volume 3274 of *LNCS*, pages 16–29. Springer-Verlag, Oct 2004.
13. I. Keider and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults. *SIGACT News*, 32(2):45–63, 2001.

14. 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.
15. A. Mostefaoui, S. Rajsbaum, and M. Raynal. Using conditions to exppedite consensus in synchronous distributed systems. In *Proc. of 17th International Conference on Ditributed Computing(DISC)*, volume 2848 of *LNCS*, pages 249–263, Oct 2003.
16. 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.
17. A. Mostéfaoui, S. Rajsbaum, and M. Raynal. The combined power of conditions and failure detectors to solve asynchronous set agreement. In *Proceedings of the 24th annual ACM symposium on Principles of distributed computing*, 2005. (to appear).
18. A. Mostéfaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based protocols for set agreement problems. In *Proc. of 17th International Conference on Ditributed Computing (DISC)*, volume 2508 of *LNCS*, pages 48–62. Springer-Verlag, 2002.
19. 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.
20. Y. Zibin. Condition-based consensus in synchronous systems. In *Proc. of 17th International Conference on Ditributed Computing(DISC)*, volume 2848 of *LNCS*, pages 239–248, Oct 2003.