Common Knowledge and Consistent Simultaneous Coordination[*]

Gil Neiger[†]
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Mark R. Tuttle
DEC Cambridge Research Lab
One Kendall Square, Building 700
Cambridge, Massachusetts 02139

July 21, 1992

Abstract

There is a very close relationship between common knowledge and simultaneity in synchronous distributed systems. The analysis of several well-known problems in terms of common knowledge has lead to round-optimal protocols for these problems, including *Reliable Broadcast*, *Distributed Consensus*, and the *Distributed Firing Squad* problem. These problems require that the correct processors coordinate their actions in some way but place no restrictions on the behavior of the faulty processors. In systems with benign processor failures, however, it is reasonable to require that the actions of a faulty processor be consistent with those of the correct processors, assuming it performs any action at all. We consider problems requiring *consistent*, *simultaneous* coordination. We then analyze these problems in terms of common knowledge in several failure models. The analysis of these stronger problems requires a stronger definition of common knowledge, and we study the relationship between these two definitions. In many cases, the two definitions are actually equivalent, and simple modifications of previous solutions yield round-optimal solutions to these problems. When the definitions differ, however, we show that such problems cannot be solved, even in failure-free executions.

[*]This paper appeared in *Distributed Computing*, 6(3):181–192, 1993. An earlier version of this paper appeared in J. van Leeuwen and N. Santoro, editors, *Proceedings of the Fourth International Workshop on Distributed Algorithms*, volume 486 of *Lecture Notes on Computer Science*, pages 334–352. Springer-Verlag, September 1990.

[†]This author was supported in part by the National Science Foundation under grants CCR-8909663 and CCR-9106627.

1 Introduction

Processor coordination and synchronization are recurring problems in fault-tolerant distributed computing. Several aspects of these problems are captured by the definition of a simultaneous choice problem given by Moses and Tuttle [17]. A *simultaneous choice problem* is one in which all of the nonfaulty processors are required to choose the same action (such as deciding on an output bit) from a set of actions and to perform that action simultaneously. Instances of simultaneous choice problems include simultaneous versions of many well-known problems, such as *Reliable Broadcast* [19], *Byzantine Agreement* [14,20], and *Distributed Firing Squad* [1,4,22]. For example, in Byzantine Agreement, each processor starts with an input bit and chooses an output bit. All nonfaulty processors must choose the same output bit and this bit must be some processor's input bit. In *Simultaneous* Byzantine Agreement [5,6], all nonfaulty processors must choose their output bit in the same round.

One property all of these problems have in common is that they define the correct behavior of the nonfaulty processors, but they do not restrict the behavior of faulty processors in any way. This is usually because problems like *Byzantine Agreement* were originally defined in the Byzantine failure model in which faulty processors can behave in completely arbitrary ways, making it impossible to require or forbid them to do anything. Subsequently, however, papers in the literature have considered the same problems in models in which processors fail in relatively benign ways. One such model is the crash failure model, in which processors fail simply by halting. In this model, it is often possible to guarantee that the faulty processors do behave consistently with the nonfaulty processors. For example, one of the earliest coordination problems is the atomic commit problem, which requires processors in a distributed database system to decide whether to commit or abort a transaction on the database. This problem guarantees that all processors (whether faulty or nonfaulty) that reach a decision to commit or abort a specific transaction must reach the same decision [9,15]. Other problems requiring that faulty processors behave consistently with nonfaulty processors appear in works by Neiger and Toueg [19] and by Gopal and Toueg [8].

Given that consistency is sometimes possible in benign failure models, it is interesting to ask whether consistency is always possible. In this paper, we define *consistent simultaneous choice* problems, in which all processors (faulty or nonfaulty) that perform an action perform the same action at the same time. We show that, in many cases, the consistent version of a simultaneous choice problem can be solved whenever the nonconsistent version can be solved, and that it can be solved without increasing the running time of the protocol.

Our approach is to analyze the state of knowledge that processors must attain in order to solve consistent simultaneous choice problems. We then to use the results of this analysis to construct *round-optimal* solutions for these problems. This general approach was introduced by Halpern and Moses [12]. They define what it means for a processor to "know" a fact and show how knowledge can be used to analyze distributed computation. In their work, the highest state of knowledge that a group of processors can attain is that of *common knowledge*. Intuitively, a fact is common knowledge if every processor knows it, every processor knows that every processor knows it, and so on.

This state of common knowledge is closely related to solving simultaneous choice problems. For example, Dwork and Moses [6] consider *Simultaneous Byzantine Agreement.* They show that, in the crash failure model, processors can choose their output bits only after certain facts have become common knowledge. They then give a protocol in which processors choose their output bits as soon as these facts become common knowledge and prove that this protocol is *optimal in all runs*: for every given context—for every pattern of failures and processor inputs—the round in which their protocol causes processors to reach agreement is at least as early as the round in which any other protocol would do so in the same context. Notice that optimal here means round-optimal.

While Dwork and Moses focus on *Simultaneous Byzantine Agreement*, their techniques apply to the general class of simultaneous choice problems. Moses and Tuttle [17] showed this by defining the class of simultaneous choice problems and by considering failure models more general than the crash model. In these new models, they again show that nonfaulty processors can perform an action simultaneously only after reaching common knowledge of certain facts, and they exhibit optimal protocols that perform an action as soon as these facts become common knowledge. Unfortunately, in the general omissions model—a model in which faulty processors may omit to send or receive messages—their optimal protocols require exponential local computation. On the other hand, they prove that testing for common knowledge is NP-hard in this model. Using the close relationship between common knowledge and simultaneity, they show that optimal protocols in this model are inherently intractable (if P$\neq$NP).

This work by Moses and Tuttle is the starting point for our own work. Like them, we begin by showing that processors can consistently perform simultaneous actions only after certain facts have become common knowledge. Our analysis, however, requires a stronger definition of common knowledge. Our definition is essentially the original definition proposed by Halpern and Moses; the definition used by Moses and Tuttle can be viewed as a subtle weakening of this definition. Surprisingly, however, we can show that these two definitions are equivalent in almost all contexts that Moses and Tuttle consider. As a result, the optimal protocols Moses and Tuttle give for simultaneous actions are (possibly after a minor modification) also optimal protocols for consistent simultaneous actions. Consequently, we show that consistency does not delay action: optimal protocols for the consistent version of a problem halt as soon as optimal protocols for the corresponding nonconsistent version of the problem. Since we show that optimal protocols for consistent problems are also optimal protocols of the corresponding nonconsistent versions, the Moses and Tuttle intractability result holds for consistent problems, too.

There is one case that Moses and Tuttle consider in which the two definitions of common knowledge are different. This is the case of the general omissions model in which as many as half of the processors can fail. In this case, we prove that no protocol solving a consistent simultaneous choice problem can cause processors to perform actions in failure-free runs, meaning that most interesting consistent simultaneous choice problems cannot be solved in this case. On the other hand, Moses and Tuttle have already shown that the nonconsistent versions of these problems can be solved and that optimal protocols exist (although these protocols are intractable). Thus, our work shows precisely where Moses and Tuttle required the added subtlety in their definition of common knowledge. When the consistent and nonconsistent simultaneous choice problems coincide, the two definitions of common knowledge are equivalent; when the problems differ, our definition of common knowledge (given by Halpern and Moses) must be weakened in order to obtain the definition (of Moses and Tuttle) most suited for nonconsistent simultaneous choice problems. Understanding the relationship between the two definitions of common knowledge is one of the primary technical contributions of our work.

This paper is organized as follows. Section 2 defines our model of computation, and Section 3 defines a processor's state of knowledge, including common knowledge. Section 4 defines consistent simultaneous choice problems, and Section 5 analyzes solutions to consistent coordination problems in terms of common knowledge and presents several optimal solutions to these problems in several variants of the omissions failure model. Section 6 shows that these problems cannot be implemented in certain systems with general omission failures. Section 7 contains some concluding remarks and mentions some open problems.

Since so many of the definitions and techniques in this paper come from Moses and Tuttle, we assume that the reader is familiar with the general definitions and results of that work, and we only sketch the points where these papers overlap. A reader intimately familiar with that work can simply read the definition of strong common knowledge in Section 3 and the definition of a consistent simultaneous choice problem in Section 4 and then skip directly to Section 5.

2 Model of a System

The systems we consider in this paper are synchronous systems of unreliable processors. Our model of computation is the model developed by Moses and Tuttle [17], which is itself derived from other models [6,12]. We sketch the model in this section and refer the reader to Moses and Tuttle for full details.

A system consists of a set $P = \{p_1, \ldots, p_n\}$ of n processors. Computation in the system proceeds in a sequence of *rounds*, with round k lasting from time $k-1$ to time k. During round k, each processor performs some local computation (and possibly some local actions), sends messages to a set of other processors, and then receives messages sent to it by other processors in that round. We think of these messages as being sent at time $k-1$ and received at time k. At any given time, a processor is in some *local state*, which we assume includes the time, the processor's identifier, and a complete history of messages it has sent and received. A processor's local state at time 0 is called an *initial state* and contains some *initial input*. A *global state* is a tuple $\langle s_1, \ldots, s_n, s_e \rangle$ of local states, one local state s_i for each processor p_i and one state s_e for the *environment*. The environment state is used to encode any information about the state of the system that cannot be deduced easily from the local states of the processors, such as the identities of the processors that have failed. A *run* is an infinite sequence of global states, typically satisfying certain obvious restrictions; for example, a processor's message history can only lengthen as time passes. Processor p_i's local state at time l in run ρ is denoted $\rho_{p_i}(l)$, or sometimes $\rho_i(l)$. A *point* is an ordered pair $\langle \rho, l \rangle$ consisting of a run ρ and a time l. A *protocol* is a (deterministic) function from a processor p_i's local state to a list of local actions it is to perform and a list of messages it is to send. A *system* is a set of runs, typically the set of all runs of a given protocol.

Processors may fail, but only two types of failures are allowed: a processor may omit to send a message its protocol requires it to send, or it may omit to receive a message sent to it by another processor. A *failure pattern* is a tuple $\pi = (\langle S_1, R_1 \rangle, \ldots, \langle S_n, R_n \rangle)$ of functions S_i and R_i mapping rounds numbers to sets of processors. The *failure pattern of a run* ρ is a failure pattern π such that, for each processor p_i and round k, (1) p_i sends no message to processors in $S_i(k)$ in round k but sends all other round k messages required by its protocol and (2) p_i receives no message from processors in $R_i(k)$ in round k but receives all other round k messages sent to it. The *input* of a run ρ is a vector $\bar{\iota} = \langle \iota_1, \ldots, \iota_n \rangle$ where ι_i is p_i's initial input in ρ. The *operating environment* of ρ is a pair $\langle \pi, \bar{\iota} \rangle$ where π and $\bar{\iota}$ are ρ's failure pattern and input, respectively. We assume that the operating environment of a run is encoded in the environment state at all times. Note that, because protocols are deterministic, a run is uniquely determined by a protocol and an operating environment.

We consider three failure models in this work:

1. the *crash* model [10,23], in which a faulty processor sends all messages required by its protocol before some round k, sends only some of the messages it is required to send in round k, and sends no messages after round k;
2. the *send omissions* model [10,16], in which a faulty processor may omit to send some of the messages required by its protocol; and
3. the *general omissions* model [17,21], in which a faulty processor may omit to send some of the messages required by its protocol and may omit to receive some of the messages sent to it by other processors.

These descriptions can be formalized in terms of failure patterns defined above. In addition to restricting the types of failures, we place a bound on the number of processors that can fail in any given run. We denote this bound by t and always assume that $n \geq t + 2$.

Since a run is uniquely determined by a protocol and an operating environment, we can compare different protocols by comparing their behaviors in the presence of the same operating environment. Two runs of two different protocols are *corresponding runs* if they have the same operating environment. We can also compare the behavior of the same protocol in slightly different operating environments. We say that two runs ρ and ρ' of a protocol $\mathcal{P}$ *differ only in* some aspect of their operating environment if and only if ρ and ρ' are the result of running $\mathcal{P}$ in operating environments that differ only (if at all) in the given aspect. For example, if ρ and ρ' differ only in that p_i sends no messages in round k of ρ', we mean that the failure patterns π and π' of ρ and ρ' are identical except that $S'_i(k) = P$ (the value of $S_i(k)$ is irrelevant). Notice that, if the messages p_i sends in round k affect messages other processors send in later rounds, then the messages sent in the two runs can be quite different. On the other hand, if $\mathcal{P}$ does not require p_i to send any messages in round k in the first place, then no processor will be able to distinguish the two runs.

3 Definitions of Knowledge

The definition of processor knowledge has become standard in distributed computing. We sketch this and related definitions in this section and refer the reader to other papers [6,12,17] for a more comprehensive review.

For the sake of exposition, we fix a particular system for the remainder of this section. We also fix a set Φ of primitive propositions together with an interpretation τ mapping each proposition $\varphi \in \Phi$ to the set $\tau(\varphi)$ of points at which φ is true. We now define a language of knowledge by closing Φ under the standard boolean connectives in the usual way and by closing under the various knowledge operators described in this section. A formula (or fact) φ in this language is either true or false at a point $\langle \rho, l\rangle$, which are denoted $\langle \rho, l\rangle \models \varphi$ or $\langle \rho, l\rangle \not\models \varphi$, respectively. A formula is *valid in the system* if it is true of all points in the system, and a formula is *valid* if it is valid in all systems.

The definition of processor knowledge is based on the idea that, given a point $\langle \rho, l\rangle$ and a processor p_i, there may be many points consistent with the information in p_i's local state at $\langle \rho, l\rangle$. We say that p_i considers a point $\langle \rho', l'\rangle$ to be *possible* at $\langle \rho, l\rangle$ if p_i has the same local state at both points. We say that p_i *knows* φ at $\langle \rho, l\rangle$, denoted $\langle \rho, l\rangle \models \mathsf{K}_i\varphi$, if φ is true at all points $\langle \rho', l'\rangle$ that p_i considers possible at $\langle \rho, l\rangle$.

The state of common knowledge plays an important role in this work. Given a fixed group G of processors, it is customary to define "*everyone in G knows φ*," denoted $\mathsf{E}_G\varphi$, by

$$\mathsf{E}_G\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in G} \mathsf{K}_i\varphi.$$

and to define "*φ is common knowledge to G*," denoted $\mathsf{C}_G\varphi$, by

$$\mathsf{C}_G\varphi \stackrel{\text{def}}{=} \mathsf{E}_G\varphi \wedge \mathsf{E}_G\mathsf{E}_G\varphi \wedge \cdots \wedge \mathsf{E}_G^m\varphi \wedge \cdots.$$

In other words, φ is common knowledge to a group G or processors if everyone knows φ, everyone knows that everyone knows φ, and so on *ad infinitum.*

This definition of common knowledge assumes that G is a fixed set. In unreliable distributed systems, however, the most interesting set is often the set $\mathcal{N}$ of nonfaulty processors, and the value of this set does not remain fixed. At any given point $\langle \rho, l\rangle$, we denote the set of processors nonfaulty in the run ρ by $\mathcal{N}(\rho, l)$. Notice that, while this set $\mathcal{N}$ does not change over time in a particular run, it does change from run to run; we define $\mathcal{N}$ to be a function of points and not runs for consistency with Moses and Tuttle [17]. Moses and Tuttle argue that the generalization of common knowledge to nonconstant sets such as $\mathcal{N}$ that vary from point to point is more subtle than simply defining

$$\mathsf{E}_{\mathcal{N}}\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in \mathcal{N}} \mathsf{K}_i\varphi.$$

They propose a generalization of $\mathsf{E}_{\mathcal{N}}\varphi$, which we denote by $\mathsf{F}_{\mathcal{N}}\varphi$ (although they continue to write $\mathsf{E}_{\mathcal{N}}\varphi$), and they argue that one must define

$$\mathsf{F}_{\mathcal{N}}\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in \mathcal{N}} \mathsf{K}_i(p_i \in \mathcal{N} \Rightarrow \varphi).$$

That is, every nonfaulty processor knows that φ is true, given that it (the processor itself) is nonfaulty. Since it is possible for p_i to be contained in $\mathcal{N}$ without knowing this, the definition of $\mathsf{F}_{\mathcal{N}}\varphi$ is weaker than $\mathsf{E}_{\mathcal{N}}\varphi$, and the definition of common knowledge based on $\mathsf{F}_{\mathcal{N}}\varphi$ is therefore weaker than the definition above based on $\mathsf{E}_{\mathcal{N}}\varphi$. For this reason, we distinguish these two definitions of common knowledge as *weak* and *strong common knowledge*, and we denote weak and strong common knowledge of φ by $\mathsf{W}_{\mathcal{N}}\varphi$ and $\mathsf{S}_{\mathcal{N}}\varphi$, respectively. (Moses and Tuttle write $\mathsf{C}_{\mathcal{N}}\varphi$ for $\mathsf{W}_{\mathcal{N}}\varphi$.) Formally,

$$\mathsf{W}_{\mathcal{N}}\varphi \stackrel{\text{def}}{=} \mathsf{F}_{\mathcal{N}}\varphi \wedge \mathsf{F}_{\mathcal{N}}\mathsf{F}_{\mathcal{N}}\varphi \wedge \cdots \wedge \mathsf{F}^m_{\mathcal{N}}\varphi \wedge \cdots$$

and, as in the definition of $\mathsf{C}_G\varphi$ above,

$$\mathsf{S}_{\mathcal{N}}\varphi \stackrel{\text{def}}{=} \mathsf{E}_{\mathcal{N}}\varphi \wedge \mathsf{E}_{\mathcal{N}}\mathsf{E}_{\mathcal{N}}\varphi \wedge \cdots \wedge \mathsf{E}^m_{\mathcal{N}}\varphi \wedge \cdots.$$

Moses and Tuttle prove that weak common knowledge is a necessary and sufficient condition for the performance of simultaneous actions, and we will prove that strong common knowledge is a necessary and sufficient condition for the performance of *consistent* simultaneous actions.

The *similarity graph* is a useful tool for thinking about weak common knowledge. The nodes of this graph are the points of the system, and there is an edge between points $\langle\rho, l\rangle$ and $\langle\rho', l\rangle$ if and only if there is some processor $p \in \mathcal{N}(\rho, l) \cap \mathcal{N}(\rho', l)$ such that $\rho_p(l) = \rho'_p(l)$. That is, there is a processor that is nonfaulty at both points and has the same local state at both points. Two points $\langle \rho, l \rangle$ and $\langle \rho', l \rangle$ are *similar*, denoted $\langle \rho, l \rangle \sim \langle \rho', l \rangle$, if they are in the same connected component of the similarity graph. The utility of the similarity graph stems from the fact that its connected components characterize the facts that are weak common knowledge:

Lemma 1: *$\langle \rho, l \rangle \models \mathsf{W}_{\mathcal{N}}\varphi$ if and only if $\langle \rho', l \rangle \models \varphi$ for all $\langle \rho', l \rangle$ satisfying $\langle \rho, l \rangle \sim \langle \rho', l \rangle$.*

For strong common knowledge, we define a *directed similarity graph*. This is a directed graph whose nodes are the points of the system and in which there is an edge from $\langle \rho, l \rangle$ to $\langle \rho', l \rangle$ if and only if there is some processor $p \in \mathcal{N}(\rho, l)$ such that $\rho_p(l) = \rho'_p(l)$. This means that there is a processor that is nonfaulty at $\langle \rho, l \rangle$ (but possibly faulty at $\langle \rho', l \rangle$) and that has the same local state at both points. Let $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$ denote the fact that $\langle \rho', l \rangle$ is reachable from $\langle \rho, l \rangle$ in the directed similarity graph. Again, it is easy to show that

Lemma 2: *$\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$ if and only if $\langle \rho', l \rangle \models \varphi$ for all $\langle \rho', l \rangle$ satisfying $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$.*

Notice that $\langle \rho, l \rangle \sim \langle \rho', l \rangle$ implies both $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$ and $\langle \rho', l \rangle \rightsquigarrow \langle \rho, l \rangle$. It follows that $\mathsf{S}_{\mathcal{N}}\varphi \Rightarrow \mathsf{W}_{\mathcal{N}}\varphi$ is valid for all facts φ. The converses of these statements do not always hold.

Weak and strong common knowledge (as well as simple knowledge) have many useful properties. For example, knowledge and weak common knowledge satisfy the axioms and rules of the modal system $S5$ [11]. Strong common knowledge also satisfies $S5$, unless the relation $\rightsquigarrow$ is not symmetric, in which case it satisfies only the axioms of the weaker modal system $S4$. We note that the results in Section 6 stem from the fact that $\rightsquigarrow$ is not symmetric in the system considered there.

Other useful properties of strong and weak common knowledge are that they satisfy the *fixed-point axioms*

- $\mathsf{W}_{\mathcal{N}}\varphi \Leftrightarrow \mathsf{F}_{\mathcal{N}}\mathsf{W}_{\mathcal{N}}\varphi$
- $\mathsf{S}_{\mathcal{N}}\varphi \Leftrightarrow \mathsf{E}_{\mathcal{N}}\mathsf{S}_{\mathcal{N}}\varphi$

and the *induction rules*

- if $\varphi \Rightarrow \mathsf{F}_{\mathcal{N}}(\varphi \wedge \psi)$ is valid in the system, then $\varphi \Rightarrow \mathsf{W}_{\mathcal{N}}\psi$ is valid in the system;
- if $\varphi \Rightarrow \mathsf{E}_{\mathcal{N}}(\varphi \wedge \psi)$ is valid in the system, then $\varphi \Rightarrow \mathsf{S}_{\mathcal{N}}\psi$ is valid in the system.

The fixed-point axioms say that, if a formula is common knowledge, then everyone knows it is common knowledge. The induction rules are frequently used as a tool to prove that a formula is common knowledge (e.g., in the proof of Lemma 3).

We end this section with the observation that, while we have defined a fact φ to be a property of points (it is either true or false at any given point), it is sometimes convenient to refer to facts as being about things other than points (such as runs). In general, a fact φ is *about* X if fixing X determines the truth (or falsity) of φ. For example, a fact φ is *about the nonfaulty processors* if fixing the set of the nonfaulty processors determines whether or not φ holds. That is, given any set $N \subseteq P$, either φ holds at all points $\langle \rho, l \rangle$ such that $\mathcal{N}(\rho, l) = N$ or at no such point. The meaning of a fact being *about the operating environment*, *about the initial states*, etc., is defined similarly.

4 Consistent Simultaneous Choices

Moses and Tuttle define *simultaneous choice problems* in which all nonfaulty processors choose the same action from a set of actions and perform it simultaneously. We define a *consistent* version of this problem in which a faulty processor must also perform the same action at the same time, if it performs any action at all.

Moses and Tuttle define a *simultaneous action* a to be an action with two associated conditions $pro(a)$ and $con(a)$, both facts about the input and the set of faulty processors, giving conditions under which a should and should not be performed. A *simultaneous choice problem* $\mathcal{C}$ is determined by a set $\{a_1, \ldots, a_m\}$ of simultaneous actions and their associated *pro* and *con* conditions, together with a failure model $\mathcal{M}$ and set $\mathcal{I}$ of initial inputs that determine a set of allowable operating environments.[1] A protocol $\mathcal{P}$ *implements* $\mathcal{C}$ if every run ρ of $\mathcal{P}$ with an allowable operating environment satisfies the following conditions:

[1]Moses and Tuttle actually allow their *pro* and *con* conditions to be facts about the operating environment in general but quickly restrict attention to facts about the input and the *existence of failures*. We note that this class of facts is slightly smaller than the class of facts about the input and *set of faulty processors*. Except for the case of the general omissions model when $n \leq 2t$, the results of Moses and Tuttle continue to hold for this larger class in all cases they consider, and these are the cases in which we use their results. Moses and Tuttle also allow processors to receive input throughout the run and not just initial input, and we could easily do the same.

1. each nonfaulty processor performs at most one of the a_j's (and does so at most once),
2. any a_j performed by some nonfaulty processor is performed *simultaneously* by all of them,
3. a_j is performed by all nonfaulty processors if ρ satisfies $pro(a_j)$, and
4. a_j is not performed by any nonfaulty processor if ρ satisfies $con(a_j)$.

It is easy to see that this specification allows the nonfaulty processors to perform an action a_j in ρ if and only if ρ satisfies the condition

$$ok_j = \neg con(a_j) \wedge \bigwedge_{k \neq j} \neg pro(a_k).$$

A faulty processor, on the other hand, is permitted to perform any action at any time. A *consistent simultaneous choice problem* restricts these faulty processors with an additional condition (a strengthening of condition 2):

2′. any a_j performed by any processor is performed *simultaneously* by all nonfaulty processors.

A consistent simultaneous choice problem $\mathcal{C}$ is *implementable* if there is some protocol that implements it.

One example of a consistent simultaneous choice problem is a consistent version of Simultaneous Byzantine Agreement. Recall that each processor begins with an input bit and chooses an output bit, subject to the condition that all output bits have the same value and that this value is the value of some processor's input bit. This problem involves a choice between two actions: a_0 corresponds to choosing 0 as the output bit, and a_1 corresponds to choosing 1. Since the value of a processor's output bit must be the value of some processor's input bit, the condition $pro(a_v)$ requiring that processors must choose the bit v is that all processors' input bits are v (we can simply define the condition $con(a_v)$ to be *false*). The enabling condition ok_v is therefore that *some* processor's input bit is v.

5 Optimal Protocols

Now let us turn our attention to optimal protocols for consistent simultaneous choice problems. A protocol $\mathcal{P}$ implementing a consistent simultaneous choice problem $\mathcal{C}$ is *optimal* (in all runs) if the following condition is satisfied [6,17]: for every protocol $\mathcal{P}'$ implementing $\mathcal{C}$ and every pair of corresponding runs of $\mathcal{P}$ and $\mathcal{P}'$, if the nonfaulty processors perform an action at time k in the run of $\mathcal{P}'$, then the nonfaulty processors perform an action no later than time k in the run of $\mathcal{P}$. Notice that this is a strong definition of optimality: whereas most definitions of an optimal protocol require that the protocol perform well in its worst case run, this definition requires that the protocol perform well in all runs.

In many cases, we can construct optimal protocols for consistent simultaneous choice problems. Our construction directly parallels those of Dwork and Moses [6] and of Moses and Tuttle [17]. The fundamental observation in these constructions is the strong relationship between common knowledge and simultaneous actions: when a consistent simultaneous action is performed, it is strong common knowledge that this action is being performed, and hence that it is enabled:

Lemma 3: *Let ρ be a run of a protocol implementing a consistent simultaneous choice $\mathcal{C}$. If an action a_j of $\mathcal{C}$ is performed by any processor at time l in ρ, then $\langle \rho, l\rangle \models \mathsf{S}_{\mathcal{N}} ok_j$.*

Proof: Let φ be the fact "a_j is being performed by some processor." Note that $\langle \rho, l\rangle \models \varphi$. We now show that $\varphi \Rightarrow \mathsf{E}_{\mathcal{N}}(\varphi \wedge ok_j)$ is valid in the system. It will then follow by the induction rule that $\varphi \Rightarrow \mathsf{S}_{\mathcal{N}} ok_j$ is also valid in the system and, thus, $\langle \rho, l\rangle \models \mathsf{S}_{\mathcal{N}} ok_j$. Let $\langle \eta, k\rangle$ be any point such that $\langle \eta, k\rangle \models \varphi$. By condition $2'$ of the definition of a consistent choice, all processors in $\mathcal{N}(\eta, k)$ execute a_j at time k in η, so $\langle \eta, k\rangle \models \mathsf{E}_{\mathcal{N}}\varphi$ (every processor performing a_j certainly knows a_j is being performed). By conditions 1, 3 and 4, $\varphi \Rightarrow (\varphi \wedge ok_j)$ is valid in the system, so $\langle \eta, k\rangle \models \mathsf{E}_{\mathcal{N}}\varphi$ implies that $\langle \eta, k\rangle \models \mathsf{E}_{\mathcal{N}}(\varphi \wedge ok_j)$ by the consequence closure axiom of $S5$ [11]. Since $\langle \eta, k\rangle$ was chosen arbitrarily, $\varphi \Rightarrow \mathsf{E}_{\mathcal{N}}(\varphi \wedge ok_j)$ is valid in the system. $\Box$

Since Lemma 3 says that no protocol can have processors perform an action until that action's enabling condition is strong common knowledge, any protocol that has processors act as soon as some enabling condition becomes strong common knowledge will be an optimal protocol. This means that we can reduce the problem of constructing optimal protocols to that of constructing protocols that cause facts to become common knowledge as soon as possible. Such protocols include the class of full-information protocols.

A *full-information protocol* [2,7,10] calls for every processor to send its entire state to all other processors in every round. Since such a protocol requires that all processors send all the information available to them in every round, it gives each processor as much information about the operating environment as any protocol could. Consequently, if a processor cannot distinguish two operating environments during runs of a full-information protocol, then the processor cannot distinguish them during runs of any other protocol (see also [6,17]):

Lemma 4 (Coan): *Let τ and τ' be runs of a full-information protocol $\mathcal{F}$, and let ρ and ρ' be runs of an arbitrary protocol $\mathcal{P}$ corresponding to τ and τ', respectively. For all processors p and times l, if $\tau_p(l) = \tau'_p(l)$, then $\rho_p(l) = \rho'_p(l)$.*

As a simple consequence, we see that facts about the operating environment become strong common knowledge during runs of full-information protocols at least as soon as they do during runs of any other protocol:

Corollary 5: *Let φ be a fact about the operating environment. Let τ and ρ be corresponding runs of a full-information protocol $\mathcal{F}$ and an arbitrary protocol $\mathcal{P}$, respectively. If $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$, then $\langle \tau, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$.*

Proof: Suppose $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$. To prove that $\langle \tau, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$, it suffices to show that $\langle \tau', l \rangle \models \varphi$ for all runs τ' of $\mathcal{F}$ such that $\langle \tau, l \rangle \rightsquigarrow \langle \tau', l \rangle$. Fix τ' and let ρ' be the corresponding run of $\mathcal{P}$. Lemma 4 and a simple inductive argument on the distance from $\langle \tau, l \rangle$ to $\langle \tau', l \rangle$ in the directed similarity graph show that $\langle \tau, l \rangle \rightsquigarrow \langle \tau', l \rangle$ implies $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$. Since $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$, we have $\langle \rho', l \rangle \models \varphi$. Since φ is a fact about the operating environment and τ' and ρ' have the same operating environment, $\langle \tau', l \rangle \models \varphi$, as desired. □

Full-information protocols are the basis of our construction of optimal protocols in the remainder of this section.

5.1 Crash and Send Omissions

Using the preceding line of reasoning (but using $\mathsf{W}_{\mathcal{N}}$ in place of $\mathsf{S}_{\mathcal{N}}$), Moses and Tuttle derive a full-information, optimal protocol $\mathcal{F}_{MT}$ for any implementable simultaneous choice $\mathcal{C}$ (see Figure 1). This protocol is *knowledge-based* since it is programmed in a language

```
repeat every round
    broadcast local state to every processor
until $\mathsf{W}_{\mathcal{N}} ok_j$ holds for some $j$;
$j \leftarrow \min\{k \mid \mathsf{W}_{\mathcal{N}} ok_k \text{ holds}\}$;
perform $a_j$;
halt.
```

Figure 1: The optimal protocol $\mathcal{F}_{MT}$

making explicit tests for common knowledge of various facts. In this protocol, each processor broadcasts its local state every round until it detects that one of the enabling actions ok_j has become weak common knowledge to the nonfaulty processors. At this point, the processor performs the action a_j, where j is the least index such that ok_j is weak common knowledge. While broadcasting local states may, in general, require sending exponentially large messages, Moses and Tuttle show that, in any of the failure models considered here, a processor's state can be encoded as a *communication graph* whose size is polynomial in n and the current round number l; thus, this protocol requires only polynomially large messages.

Fully implementing $\mathcal{F}_{MT}$ requires implementing the tests for weak common knowledge that appear in $\mathcal{F}_{MT}$. A satisfactory test should accept a nonfaulty processor's local state at a point as input and determine whether the formula is weak common knowledge at that point. In general, given any fact ψ—such as $\mathsf{W}_{\mathcal{N}}\varphi$—and any set S of processors—such as the set $\mathcal{N}$ of nonfaulty processors—a *test for ψ for S* is an algorithm A that

1. accepts as input the local state of any processor at any point in the system and returns either *true* or *false*, and

2. given the local state $\rho_i(l)$ of a processor p_i in S at a point $\langle \rho, l\rangle$ in the system, returns *true* if and only if $\langle \rho, l\rangle \models \psi$.

Moses and Tuttle construct tests for $\mathsf{W}_{\mathcal{N}}\varphi$ for the set $\mathcal{N}$ of nonfaulty processors. Their tests for $\mathsf{W}_{\mathcal{N}}\varphi$ make use of tests for φ, and their tests for $\mathsf{W}_{\mathcal{N}}\varphi$ run in polynomial time if and only if the tests for φ do so in the first place. Consequently, Moses and Tuttle define a class of *practical* simultaneous choice problems, for which all conditions ok_i can be tested in polynomial time, and they argue that every natural simultaneous choice problem is practical.

Moses and Tuttle make a rather surprising observation that, in the crash and send omissions models, even the faulty processors can use their tests for common knowledge: given the local state of *any* processor p_i (faulty or nonfaulty) at $\langle \rho, l \rangle$, their test for $\mathsf{W}_{\mathcal{N}}\varphi$ returns *true* if and only if $\langle \rho, l \rangle \models \mathsf{W}_{\mathcal{N}}\varphi$. Thus, their test is a test for $\mathsf{W}_{\mathcal{N}}\varphi$ for the set P of *all* processors. We refer to such a test simply as a *test for* $\mathsf{W}_{\mathcal{N}}\varphi$. Thus, Moses and Tuttle effectively show the following:

Theorem 6 (Moses and Tuttle): *If C is an implementable, consistent simultaneous choice in the send omissions model, then $\mathcal{F}_{MT}$ is an optimal protocol for C. Furthermore, if C is practical, then $\mathcal{F}_{MT}$ can be implemented in polynomial time.*

5.2 General Omissions with $n > 2t$

According to Lemma 3, strong common knowledge is a necessary condition for consistently performing simultaneous actions, and yet the protocol $\mathcal{F}_{MT}$ is programmed using tests for weak common knowledge. This is because the observation made by Moses and Tuttle, that both faulty and nonfaulty processors can follow their tests for $\mathsf{W}_{\mathcal{N}}\varphi$, implies that weak common knowledge to the nonfaulty processors is equivalent to weak common knowledge to *all* processors: that is, $\mathsf{W}_{\mathcal{N}}\varphi \Leftrightarrow \mathsf{W}_{P}\varphi$ for all facts φ. From this, it follows that $\mathsf{W}_{\mathcal{N}}\varphi \Rightarrow \mathsf{S}_{\mathcal{N}}\varphi$, since $\mathsf{W}_{P}\varphi$ clearly implies $\mathsf{S}_{\mathcal{N}}\varphi$. Since we already observed that $\mathsf{S}_{\mathcal{N}}\varphi \Rightarrow \mathsf{W}_{\mathcal{N}}\varphi$, we have $\mathsf{W}_{\mathcal{N}}\varphi \Leftrightarrow \mathsf{S}_{\mathcal{N}}\varphi$.

It turns out that the same is true in the general omissions model, but only when $n > 2t$. To prove this fact, it is enough to show that the relations $\sim$ and $\rightsquigarrow$ are the same. This proof makes use of the notion of two runs ρ and ρ' of a protocol $\mathcal{P}$ differing only in some aspect of their operating environment. Remember, for example, that, if ρ and ρ' differ only in that p_i sends no messages in round k, we mean that the failure patterns π and π' of ρ and ρ' are identical except that $S_i'(k) = P$, regardless of the value of $S_i(k)$.

Lemma 7: *Consider any system in the general omissions model with $n > 2t$. If $\langle \rho, l \rangle$ and $\langle \eta, l \rangle$ are two points of the system, then $\langle \rho, l \rangle \rightsquigarrow \langle \eta, l \rangle$ if and only if $\langle \rho, l \rangle \sim \langle \eta, l \rangle$.*

Proof: Since $\langle \rho, l\rangle \sim \langle \eta, l\rangle$ implies $\langle \rho, l\rangle \rightsquigarrow \langle \eta, l\rangle$, we prove only that $\langle \rho, l\rangle \rightsquigarrow \langle \eta, l\rangle$ implies $\langle \rho, l\rangle \sim \langle \eta, l\rangle$. It suffices to show that, if $\rho_p(l) = \eta_p(l)$ for some $p \in \mathcal{N}(\rho, l)$, then $\langle \rho, l\rangle \sim \langle \eta, l\rangle$.

Suppose $\rho_p(l) = \eta_p(l)$ for some $p \in \mathcal{N}(\rho, l)$. Let $A = \mathcal{N}(\rho, l)$ and $B = \mathcal{N}(\eta, l)$. If $p \in B$, then p is nonfaulty at both points and, thus, $\langle \rho, l\rangle \sim \langle \eta, l\rangle$; suppose instead $p \in A - B$. Let $N = A \cap B$, and notice that N is the set of processors nonfaulty in both runs. Since all processors faulty in one or both runs are in $P - N$, we have $|P - N| \leq 2t$. Thus, $N \neq \emptyset$ since $n > 2t$. Let q be a processor in N.

We can assume without loss of generality that no processors fail after time l in ρ and η—that all processors send and receive all messages in all rounds after time l—since messages sent after time l do not affect processor states at time l. Let $\hat{\rho}$ and $\hat{\eta}$ be runs differing from ρ and η, respectively, only in that processors in $P - \{p\}$ also receive all messages from all processors in round l of $\hat{\rho}$ and $\hat{\eta}$. Note that A is contained in the set of nonfaulty processors in both ρ and $\hat{\rho}$ and that B is contained in the set of nonfaulty processors in η and $\hat{\eta}$. Note also that p has the same local state at time l in the four runs ρ, $\hat{\rho}$, $\hat{\eta}$, and η. It is clear that $\langle \rho, l\rangle \sim \langle \hat{\rho}, l\rangle$, since p is nonfaulty in both runs and $\rho_p(l) = \hat{\rho}_p(l)$. To see that $\langle \eta, l\rangle \sim \langle \hat{\eta}, l\rangle$, let η' be a run differing from η only in that processors in $P - \{p, q\}$ receive all messages from all processors in round l of η'. Note that $\langle \eta, l\rangle \sim \langle \eta', l\rangle$ since q is nonfaulty in both runs and has the same local state at both points. Similarly, note that $\langle \eta', l\rangle \sim \langle \hat{\eta}, l\rangle$ since there is a processor r distinct from p and q that is nonfaulty in η' (hence in $\hat{\eta}$) and has the same local state at both points (r must exist since we assume $n \geq t + 2$). Thus, $\langle \eta, l\rangle \sim \langle \hat{\eta}, l\rangle$.

We claim that there are no missing messages among processors in A in $\hat{\eta}$. Since A is contained in the set $\mathcal{N}(\hat{\rho}, l)$ of nonfaulty processors in $\hat{\rho}$, there are no missing messages between processors in A in $\hat{\rho}$. In particular, p's state at $\langle \hat{\rho}, l\rangle$ records the fact that there are no missing messages between processors in A for the first $l - 1$ rounds of $\hat{\rho}$ and that there are no missing messages from processors in A to p in round l. Since p has the same local state at $\langle \hat{\eta}, l\rangle$, the same is true of $\hat{\eta}$. Furthermore, since all processors in $P - \{p\}$ receive messages from all processors in round l of $\hat{\eta}$ (and in particular from processors in A), there are no missing messages between processors in A in $\hat{\eta}$. This proves our claim.

Since there are no missing messages among processors in A in $\hat{\eta}$, every missing message in $\hat{\eta}$ must have at least one processor in $P - A$ as a sender or receiver. This means that it is consistent with the pattern of missing messages in $\hat{\eta}$ that $P - A$ contains the set of processors faulty in $\hat{\eta}$. Consequently, since $|P - A| \leq t$, there is a run $\hat{\eta}'$ indistinguishable from $\hat{\eta}$—the same messages are sent and processors have the same local states—except that A is contained in the set of nonfaulty processors in $\hat{\eta}'$ instead of B.

Since q is nonfaulty in both runs $\hat{\eta}$ and $\hat{\eta}'$—note that $q \in N = A \cap B \subseteq \mathcal{N}(\hat{\eta}, l) \cap \mathcal{N}(\hat{\eta}', l)$—and certainly has the same local state at both points, $\langle \hat{\eta}, l \rangle \sim \langle \hat{\eta}', l \rangle$. Note, furthermore, that p is nonfaulty in both $\hat{\rho}$ and $\hat{\eta}'$ since $p \in A$, and that p also has the same local state at $\langle \hat{\eta}', l \rangle$ and $\langle \hat{\rho}, l \rangle$, so $\langle \hat{\eta}', l \rangle \sim \langle \hat{\rho}, l \rangle$. By the transitivity of the similarity relation, therefore, we have $\langle \rho, l \rangle \sim \langle \eta, l \rangle$. □

It follows from Lemma 7 that strong and weak common knowledge are equivalent with respect to the nonfaulty processors:

Theorem 8: *Consider any system in the general omissions model with $n > 2t$. The formula $\mathsf{W}_{\mathcal{N}}\varphi \Leftrightarrow \mathsf{S}_{\mathcal{N}}\varphi$ is valid in the system for every fact φ.*

Proof: It suffices to show that $\mathsf{W}_{\mathcal{N}}\varphi \Rightarrow \mathsf{S}_{\mathcal{N}}\varphi$ is valid in the system, since we already noted in Section 3 that $\mathsf{S}_{\mathcal{N}}\varphi \Rightarrow \mathsf{W}_{\mathcal{N}}\varphi$ is valid. Suppose $\langle \rho, l \rangle \models \mathsf{W}_{\mathcal{N}}\varphi$. Lemma 2 implies that to prove that $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$ we need only show that $\langle \rho', l \rangle \models \varphi$ for all $\langle \rho', l \rangle$ such that $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$. Since, by Lemma 7, $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$ implies $\langle \rho, l \rangle \sim \langle \rho', l \rangle$ and, since $\langle \rho, l \rangle \models \mathsf{W}_{\mathcal{N}}\varphi$ implies $\langle \rho', l \rangle \models \varphi$ for all $\langle \rho', l \rangle$ such that $\langle \rho, l \rangle \sim \langle \rho', l \rangle$ by Lemma 1, we have $\langle \rho', l \rangle \models \varphi$ for all $\langle \rho', l \rangle$ such that $\langle \rho, l \rangle \rightsquigarrow \langle \rho', l \rangle$. Thus, $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\varphi$. □

Moses and Tuttle prove that $\mathcal{F}_{MT}$ is an optimal protocol for simultaneous choice problems in the general omissions model. For practical problems, they show how to implement tests for $\mathsf{W}_{\mathcal{N}}\varphi$ for the nonfaulty processors in polynomial space, although it is not possible to do so in polynomial time (if P≠NP). The fact that weak and strong common knowledge are equivalent (when $n > 2t$) suggests that $\mathcal{F}_{MT}$ might also be an optimal protocol for *consistent* simultaneous choice problems in the general omissions model, as well as in the crash and send omissions models. But, while the nonfaulty processors can test for strong common knowledge of ok_j, it is not immediately clear that faulty processors can do the same. This raises problems such as the following: (1) the nonfaulty processors will know ok_1 and ok_2 are strong common knowledge while the faulty processors know only that ok_2 is strong common knowledge, resulting in the nonfaulty processors performing a_1 and the faulty processors performing a_2; and (2) the nonfaulty processors will know ok_1 is strong common knowledge at time k while the faulty processors will not know ok_1 is strong common knowledge until time $k+1$, resulting in the nonfaulty processors performing a_1 at time k and the faulty ones at time $k+1$. Fortunately, we can use Lemma 7 again to prove that every processor—faulty or nonfaulty—knows the truth of $\mathsf{S}_{\mathcal{N}}\varphi$, provided that this processor does not know that it is faulty (that is, its local state does not prove it must be faulty):

Lemma 9: *Consider any system in the general omissions model with $n > 2t$. The formula $\neg\mathsf{K}_i(p_i \textit{ is faulty}) \Rightarrow \mathsf{K}_i(\mathsf{S}_{\mathcal{N}}\varphi) \vee \mathsf{K}_i(\neg\mathsf{S}_{\mathcal{N}}\varphi)$ is valid in the system for every processor p_i and every fact φ.*

Proof: Suppose $\langle\rho,k\rangle \models \neg\mathsf{K}_i(p_i \text{ is faulty})$. We prove that $\langle\rho,k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$ implies $\langle\rho,k\rangle \models \mathsf{K}_i(\mathsf{S}_{\mathcal{N}}\varphi)$, and an analogous argument proves that $\langle\rho,k\rangle \models \neg\mathsf{S}_{\mathcal{N}}\varphi$ implies $\langle\rho,k\rangle \models \mathsf{K}_i(\neg\mathsf{S}_{\mathcal{N}}\varphi)$.

To show that $\langle\rho,k\rangle \models \mathsf{K}_i(\mathsf{S}_{\mathcal{N}}\varphi)$, it suffices to show that $\langle\rho',k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$ for every point $\langle\rho',k\rangle$ such that p_i has the same local state at $\langle\rho,k\rangle$ and $\langle\rho',k\rangle$. Given such a point $\langle\rho',k\rangle$, by Lemma 2, it is enough to show that $\langle\rho'',k\rangle \models \varphi$ for every point $\langle\rho'',k\rangle$ such that $\langle\rho',k\rangle \rightsquigarrow \langle\rho'',k\rangle$. Fix such a point $\langle\rho'',k\rangle$. Since $\langle\rho,k\rangle \models \neg\mathsf{K}_i(p_i \text{ is faulty})$, there exists a point $\langle\eta,k\rangle$ such that p_i is nonfaulty in η and has the same local state at both $\langle\rho,k\rangle$ and $\langle\eta,k\rangle$ and, hence, also at $\langle\rho',k\rangle$ and $\langle\eta,k\rangle$; in other words, $\langle\eta,k\rangle \rightsquigarrow \langle\rho,k\rangle$ and $\langle\eta,k\rangle \rightsquigarrow \langle\rho',k\rangle$. By Lemma 7, $\langle\eta,k\rangle \rightsquigarrow \langle\rho,k\rangle$ implies $\langle\eta,k\rangle \sim \langle\rho,k\rangle$, which implies $\langle\rho,k\rangle \rightsquigarrow \langle\eta,k\rangle$. It follows that

$$\langle\rho,k\rangle \rightsquigarrow \langle\eta,k\rangle \rightsquigarrow \langle\rho',k\rangle \rightsquigarrow \langle\rho'',k\rangle,$$

and hence, by Lemma 2, that $\langle\rho'',k\rangle \models \varphi$ since $\langle\rho,k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$. □

One consequence of this result is that any test for $\mathsf{S}_{\mathcal{N}}\varphi$ for the set of nonfaulty processors is *almost* a test for $\mathsf{S}_{\mathcal{N}}\varphi$ for the set of *all* processors: such a test correctly computes whether $\mathsf{S}_{\mathcal{N}}\varphi$ holds when applied to the local state of *any* processor—faulty or nonfaulty—provided the processor's local state does not *prove* that it must be faulty.

Corollary 10: *Consider any system in the general omissions model with $n > 2t$. Let A_φ be a test for $\mathsf{S}_{\mathcal{N}}\varphi$ for the nonfaulty processors. If $\langle \rho, k\rangle \models \neg\mathsf{K}_i(p_i$ is faulty), then A_φ on input $\rho_i(k)$ returns true if and only if $\langle \rho, k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$.*

Proof: Since $\langle \rho, k\rangle \models \neg\mathsf{K}_i(p_i$ is faulty), there is a point $\langle \eta, k\rangle$ such that p_i is nonfaulty at $\langle \eta, k\rangle$ and has the same local state at $\langle \rho, k\rangle$ and $\langle \eta, k\rangle$. By Lemma 9, processor p_i knows the truth of $\mathsf{S}_{\mathcal{N}}\varphi$ at $\langle \rho, k\rangle$, so $\mathsf{S}_{\mathcal{N}}\varphi$ holds at $\langle \rho, k\rangle$ if and only if $\mathsf{S}_{\mathcal{N}}\varphi$ holds at $\langle \eta, k\rangle$. Since the test A_φ must return the same answer when given p_i's state at these two points as input (the local states are the same), and since A_φ returns true at $\langle \eta, k\rangle$ if and only if $\langle \eta, k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$ (A_φ is a test for the nonfaulty processors), it follows that A_φ returns true at $\langle \rho, k\rangle$ if and only if $\langle \rho, k\rangle \models \mathsf{S}_{\mathcal{N}}\varphi$. □

In particular, consider the polynomial-space tests for $\mathsf{W}_{\mathcal{N}} ok_j$ for the nonfaulty processors given by Moses and Tuttle. By Theorem 8, these are also tests for $\mathsf{S}_{\mathcal{N}} ok_j$ for the nonfaulty processors. In fact, they are accurate tests for $\mathsf{S}_{\mathcal{N}} ok_j$ when given the local state of any processor whose state does not prove that it is faulty. Notice that a processor p_i knows it is faulty (its local state proves it is faulty) if and only if it is faulty in every failure pattern consistent with the messages recorded in its local state as missing, and this can also be checked in polynomial space. Thus, the protocol $\mathcal{F}'_{MT}$ given in Figure 2 is an optimal protocol for a consistent choice running in polynomial space.

Theorem 11: *If $\mathcal{C}$ is an implementable, consistent simultaneous choice in the general omissions model with $n > 2t$, then $\mathcal{F}'_{MT}$ is an optimal protocol for $\mathcal{C}$. Furthermore, if $\mathcal{C}$ is practical, then $\mathcal{F}'_{MT}$ can be implemented in polynomial space.*

Proof: Let $\mathcal{P}$ be any protocol implementing $\mathcal{C}$, and let τ and ρ be corresponding runs of $\mathcal{F}'_{MT}$ and $\mathcal{P}$. We will prove that τ satisfies the four conditions in the definition of a consistent simultaneous choice problem given in Section 4. First, however, it is convenient to prove

```
$nonfaulty \leftarrow true$
repeat every round
    broadcast local state to every processor;
    if $\mathsf{K}_i(p_i$ is faulty$)$ then $nonfaulty \leftarrow false$
until $nonfaulty$ and $\mathsf{S}_{\mathcal{N}}\, ok_j$ holds for some $j$;
$j \leftarrow \min\{k \mid \mathsf{S}_{\mathcal{N}}\, ok_k$ holds$\}$;
perform $a_j$;
halt.
```

Figure 2: The optimal protocol $\mathcal{F}'_{MT}$

the following claim: if some processor performs an action at time l in ρ, then all nonfaulty processors perform an action no later than time l in τ. Suppose that some processor performs action a_j in ρ at time l. Lemma 3 implies that $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}}\, ok_j$. By Corollary 5, $\langle \tau, l \rangle \models \mathsf{S}_{\mathcal{N}}\, ok_j$. From Figure 2, it is clear that the nonfaulty processors perform some action no later than time l in τ. This proves the claim. It will follow from this claim that $\mathcal{F}'_{MT}$ is an optimal protocol for $\mathcal{C}$, once we have shown that $\mathcal{F}'_{MT}$ solves $\mathcal{C}$ (that τ satisfies the four conditions in Section 4):

1. Each processor clearly performs at most one of the a_j's in τ and does so at most once.

2′. Suppose that some processor p performs some action a_j at time l in τ. Then p does not know it is faulty at any time through time l, and hence the tests p has been following for $\mathsf{S}_{\mathcal{N}}\, ok_k$ have been accurate through time l by Corollary 10. Consequently, this must be the first round that any fact ok_k has been strong common knowledge. Furthermore, j must be the least index among the indices k of the facts ok_k that are strong common knowledge at time l. Since the tests followed by nonfaulty processors are always accurate, they reach the same conclusions at time l and perform a_j at time l in τ.

3. Suppose τ satisfies $pro(a_j)$. Since τ and ρ are corresponding runs and $pro(a_j)$ is a fact about the environment, the run ρ must also satisfy $pro(a_j)$. Since $\mathcal{P}$ implements $\mathcal{C}$, the action a_j is performed in ρ at some time l. By the above claim, the nonfaulty processors perform some action no later than time l in τ. This action, however, must also be a_j, since the fact that τ satisfies $pro(a_j)$ implies that τ does not satisfy ok_k for any $k \neq j$ and hence that no such condition can become strong common knowledge during τ.

4. Suppose τ satisfies $con(a_j)$. Then τ does not satisfy ok_j, so ok_j will never become strong common knowledge in τ, and a_j will never be performed in τ.

Finally, to see that $\mathcal{F}'_{MT}$ can be implemented in polynomial space, assuming $\mathcal{C}$ is practical, notice that this is already true for $\mathcal{F}_{MT}$. The only modification made to $\mathcal{F}_{MT}$ to obtain $\mathcal{F}'_{MT}$ is to add a test for whether a processor knows itself to be faulty. As noted earlier, this test can be performed in polynomial space. □

Moses and Tuttle prove that testing for weak common knowledge is NP-hard in the general omissions model when $n > 2t$. The same is true for strong common knowledge, since the two definitions are equivalent. Consequently, the protocol $\mathcal{F}'_{MT}$ cannot be implemented in polynomial time in this model (if P$\neq$NP). In fact, Moses and Tuttle prove that any optimal protocol for general simultaneous choice problems requires processors to perform NP-hard computations, and that optimal protocols are thus inherently intractable. A reduction similar to that given by Moses and Tuttle shows that the same is true of optimal protocols for consistent simultaneous choice problems.

6 An Impossibility Result

While weak and strong common knowledge are equivalent in the general omissions model when $n > 2t$, they are *not* equivalent when $n \leq 2t$. It is not hard to show that some fact about the initial state must become weak common knowledge by time $t + 1$ at the latest in any run [6,17]. This section shows that, in many runs, such facts *never* become strong common knowledge; this is true even in runs in which no failures occur. This implies that many consistent simultaneous choice problems cannot be implemented in these systems. For example, no problem whose *pro* conditions are nontrivial facts about the input (and not about the set of faulty processors) can be implemented in these systems. Consequently, the impossibility of solving *Uniform Agreement* in this context, as shown by Neiger and Toueg [19], follows as a corollary of this result.

This distinction between consistent and nonconsistent simultaneous choice problems comes about because of the fact that, when $n \leq 2t$, the system can become *partitioned.* There can be two disjoint sets of at most t processors such that processors within a set communicate with no trouble, but processors in different sets never communicate. Since this behavior is consistent with either set of processors being the set of faulty processors, no processor can know whether or not it is faulty. In the context of nonconsistent simultaneous choices, this is not a problem, since the behavior of the faulty processors is unimportant, so each set can simply behave as if it is the set of nonfaulty processors. In the context of *consistent* choices, however, this behavior is critical: because the processors may be isolated from important information and not know whether or not they are faulty, the correct processors can become "paralyzed" and unable to act.

To make this precise, we say that two sets A and B *partition* the set of processors if A and B are two nonempty, disjoint sets of processors such that $A \cup B = P$ and $|A|, |B| \leq t$. Given a failure-free run ρ, let ρ_k be the run identical to ρ (the failure pattern is the same) except that, for every round $l > k$, no processor in B sends or receives a message to or from any processor in A in round l of ρ_k; note that A is the set of nonfaulty processors in ρ_k. We say that ρ_k is *the result of partitioning ρ into A and B from time k.* The following lemma says that $\langle \rho, l \rangle \rightsquigarrow \langle \rho_k, l \rangle$ and $\langle \rho_k, l \rangle \rightsquigarrow \langle \rho, l \rangle$ for every $k \geq 0$, and in particular for $k = 0$.

Lemma 12: *Consider any system in the general omissions model with $n \leq 2t$, and let ρ be any failure-free run of this system. Suppose A and B partition the set of processors, and suppose ρ_k is the result of partitioning ρ into A and B from time k. Then $\langle \rho, l \rangle \rightsquigarrow \langle \rho_k, l \rangle$ and $\langle \rho_k, l \rangle \rightsquigarrow \langle \rho, l \rangle$ for every $k \geq 0$.*

Proof: The proof is by reverse induction on k. For the base case of $k = l$, the result is trivially true since every processor has the same local state at time l in ρ and ρ_l. In this case, we actually have $\langle \rho, l \rangle \sim \langle \rho_l, l \rangle$.

For $k < l$, suppose the inductive hypothesis holds for $k+1$; that is, $\langle \rho, l \rangle \rightsquigarrow \langle \rho_{k+1}, l \rangle$ and $\langle \rho_{k+1}, l \rangle \rightsquigarrow \langle \rho, l \rangle$. Let η_1 be a run differing from ρ_{k+1} only in that no processor in B receives a message from any processors in A in round $k+1$ of η_1. Note that A is the set of nonfaulty processors in both runs and that every nonfaulty processor has the same local state at time l in both runs, so $\langle \rho_{k+1}, l \rangle \rightsquigarrow \langle \eta_1, l \rangle$ and $\langle \eta_1, l \rangle \rightsquigarrow \langle \rho_{k+1}, l \rangle$. Since the only missing messages in η_1 are between processors in A and processors in B, and since both A and B are of size at most t, the behavior in η_1 is consistent with either A or B being the set of faulty processors. Let η_2 be a run indistinguishable from η_1—the same messages are sent and processors have the same local states—except that now A is the set of faulty processors and B is the set of nonfaulty processors. Since every nonfaulty processor in η_1 has the same local state at time l in the two runs, $\langle \eta_1, l\rangle \rightsquigarrow \langle \eta_2, l\rangle$. Similarly, since every nonfaulty processor in η_2 has the same local state at time l in the two runs, $\langle \eta_2, l\rangle \rightsquigarrow \langle \eta_1, l\rangle$.[2] Now let η_3 be a run differing from η_2 only in that no (faulty) processor in A receives a message from any processor in B in round $k+1$ of η_3. Because the set of nonfaulty processors is the same in η_2 and η_3, and because every nonfaulty processor has the same local state at time l in both runs, we have $\langle \eta_2, l\rangle \rightsquigarrow \langle \eta_3, l\rangle$ and $\langle \eta_3, l\rangle \rightsquigarrow \langle \eta_2, l\rangle$. Note that η_3 is the result of partitioning ρ into B and A from time k, while the desired ρ_k is the result of partitioning ρ into A and B from time k; that is, the only difference between the two runs is that A is the set of nonfaulty processors in ρ_k, while B is the set of nonfaulty processors in η_3. On the other hand, since every processor (faulty or nonfaulty) has the same local state at time l in both runs, it is clear that $\langle \eta_3, l\rangle \rightsquigarrow \langle \rho_k, l\rangle$ and $\langle \rho_k, l\rangle \rightsquigarrow \langle \eta_3, l\rangle$. It follows by transitivity that $\langle \rho, l\rangle \rightsquigarrow \langle \rho_k, l\rangle$ and $\langle \rho_k, l\rangle \rightsquigarrow \langle \rho, l\rangle$, as desired. □

Using Lemma 12, we can now show that, even in failure-free runs, no fact about the input and the set of faulty processors—and thus no enabling condition ok_j (a fact about the input and set of faulty processors)—can become strong common knowledge.

Lemma 13: *Consider any system in the general omissions model with $n \leq 2t$. Let φ be a fact about the input and the set of faulty processors that is not valid in the system. If $\langle \rho, l\rangle$ is any point of any failure-free run ρ of the system, then $\langle \rho, l\rangle \not\models \mathsf{S}_{\mathcal{N}}\varphi$.*

Proof: Let $\bar{\imath}$ be the input to ρ. Since φ is a fact about the input and the set of faulty processors that is not valid in the system, there is an input vector $\bar{\imath}' \in \mathcal{I}^n$ and a set $N' \subseteq P$ such that $\langle \rho', l\rangle \not\models \varphi$ for any run ρ' with input $\bar{\imath}'$ and with N' as its set of nonfaulty

[2]Note that we do not have $\langle \eta_1, l\rangle \sim \langle \eta_2, l\rangle$, since there is no processor that is nonfaulty in both runs. For this reason, the proof does not hold when $\rightsquigarrow$ is replaced by $\sim$ and the consequences of this lemma thus apply only to strong common knowledge and not to weak common knowledge.

processors. Two input vectors $\bar{\iota}_a$ and $\bar{\iota}_b$ are *adjacent*—denoted $\bar{\iota}_a \leftrightarrow \bar{\iota}_b$—if they differ on the initial state of only one processor; that is, for some processor p_i, we have $\iota_{j,a} = \iota_{j,b}$ for all $j \neq i$. It is not hard to see that for some $m \geq 0$ there are vectors $\bar{\iota}_0, \bar{\iota}_1, \ldots, \bar{\iota}_m$ such that $\bar{\iota} = \bar{\iota}_0 \leftrightarrow \bar{\iota}_1 \leftrightarrow \cdots \leftrightarrow \bar{\iota}_m = \bar{\iota}'$. For every j, let ρ_j be the failure-free run with input $\bar{\iota}_j$ (thus, $\rho = \rho_0$).

To complete the proof, it is enough to show that $\langle \rho, l \rangle \rightsquigarrow \langle \rho_j, l \rangle$ for every $j \geq 0$. In particular, suppose $\langle \rho, l \rangle \rightsquigarrow \langle \rho_m, l \rangle$. Let ρ'_m be the run differing from ρ_m only in that no processor in $P - N'$ receives any message in round l of ρ'_m. Notice that the number of processors that fail in ρ'_m is at most t since N' is the set of nonfaulty processors in ρ' above, and hence that ρ'_m is a legitimate run. Since $\bar{\iota}'$ is the input to ρ'_m and N' is the set of nonfaulty processors in ρ'_m, we have $\langle \rho'_m, l \rangle \not\models \varphi$. Furthermore, since processors in N' have the same local state at time l in both runs, we have $\langle \rho_m, l \rangle \rightsquigarrow \langle \rho'_m, l \rangle$. It follows that $\langle \rho, l \rangle \rightsquigarrow \langle \rho'_m, l \rangle$ and yet $\langle \rho'_m, l \rangle \not\models \varphi$; thus, $\langle \rho, l \rangle \not\models \mathsf{S}_{\mathcal{N}}\varphi$ by Lemma 2.

We proceed by induction on j to show that $\langle \rho, l \rangle \rightsquigarrow \langle \rho_j, l \rangle$ for every $j \geq 0$. For $j = 0$, we are done since $\rho = \rho_0$. For $j > 0$, suppose the inductive hypothesis holds for $j - 1$; that is, $\langle \rho, l \rangle \rightsquigarrow \langle \rho_{j-1}, l \rangle$. By definition, ρ_{j-1} and ρ_j differ only in the input to some processor p. Consider any partition of P into sets A and B with $p \in B$, and let η_{j-1} and η_j be the result of partitioning ρ_{j-1} and ρ_j, respectively, into A and B from time 0. By Lemma 12, we have $\langle \rho_{j-1}, l \rangle \rightsquigarrow \langle \eta_{j-1}, l \rangle$ and $\langle \eta_j, l \rangle \rightsquigarrow \langle \rho_j, l \rangle$. Because η_{j-1} and η_j differ only in the input to p, and because there is no message from $p \in B$ to any processor in A in either run, all (nonfaulty) processors in A have the same local state at time l in both runs, and we have $\langle \eta_{j-1}, l \rangle \rightsquigarrow \langle \eta_j, l \rangle$. It follows that $\langle \rho, l \rangle \rightsquigarrow \langle \rho_j, l \rangle$, as desired. $\square$

From Lemma 13, it is clear that strong common knowledge of nonvalid facts cannot be achieved in failure-free runs. A consistent simultaneous choice problem is *nontrivial* if none of its enabling conditions ok_j are valid. It now follows that there is no solution to any nontrivial consistent simultaneous choice in this model:

Theorem 14: *In the general omissions model with $n \leq 2t$, if $\mathcal{P}$ implements a nontrivial consistent simultaneous choice problem $\mathcal{C}$, then no action is performed in failure-free runs of*

$\mathcal{P}$.

Proof: Suppose by way of contradiction that some processor performs an action a_j at time l in a failure-free run ρ of of $\mathcal{P}$. Since $\mathcal{P}$ implements $\mathcal{C}$, Lemma 3 says that $\langle \rho, l \rangle \models \mathsf{S}_{\mathcal{N}} ok_j$. But Lemma 13 says that this is impossible, since ok_j is a nonvalid fact about the input and set of faulty processors. It follows that no action is performed in failure-free runs of $\mathcal{P}$ after all. □

Theorem 14 states that consistent simultaneous choices cannot be implemented in systems with general omission failures in which half or more of the processors can fail. This is in marked contrast to the results of Section 5.2, which show that such choices can be implemented in systems in which fewer than half of the processors can fail.

7 Conclusions

In this paper, we have studied the existence of optimal solutions to consistent simultaneous choice problems under several failure models. Consistent simultaneous choice problems are simultaneous choice problems in which a faulty processor that performs an action must perform the same action as the nonfaulty processors and at the same time. In the crash and send omissions failure models, optimal protocols for simultaneous choice problems derived elsewhere [6,17] are also optimal protocols for consistent simultaneous choice problems and run in polynomial time. We have shown that, in the general omissions failure model in which fewer than half the processors can fail (that is, in which $n > 2t$), a modification of these optimal protocols are optimal protocols in this model as well. These protocols require exponential time, but any optimal protocol in this model is inherently intractable if P$\neq$NP (it requires processors to perform NP-hard computations). Furthermore, in all of these models, the optimal protocols for the consistent version of a simultaneous choice problem halt at precisely the same time as the optimal protocols for the original version. Thus, we can obtain consistency in these models at no extra cost. Finally, we have shown that, in the general omissions failure model in which half or more of the processors may fail (that is, in which $n \leq 2t$), there is no solution to any nontrivial consistent simultaneous choice problem: processors cannot consistently coordinate actions even in failure-free runs.

One of the technical contributions of this work is an exploration of the relationship between the definition of common knowledge used by Moses and Tuttle [17] and the original definition proposed by Halpern and Moses [12]. Moses and Tuttle define weak common knowledge, and Halpern and Moses essentially define strong common knowledge (although their definition is formulated in terms of fixed sets and not nonconstant sets such as the set of nonfaulty processors). Because we have shown that the two definitions are equivalent in most of the cases considered by Moses and Tuttle, we have shown that their work could have been simplified by using the simpler definition of common knowledge originally proposed by Halpern and Moses. On the other hand, where we have shown the two definitions are different (in the general omissions model with $n \leq 2t$), consistent simultaneous choice problems cannot be solved, whereas Moses and Tuttle have shown that the original simultaneous choice problems *can* be solved. In this sense, our work shows precisely where the subtlety in the definition of weak common knowledge is required by the work of Moses and Tuttle. It also shows that the difference between strong and weak common knowledge is at the heart of the difference between the consistent and original versions of simultaneous choice problems.

In this work we have analyzed problems requiring simultaneous coordination of actions, but nonsimultaneous coordination is also of interest. For example, Halpern, Moses, and Waarts [13] have considered *Eventual Byzantine Agreement*, a problem in which correct processors must agree on the value of their output bits but need not choose these bits at the same time. They show that solving such problems requires a variant of common knowledge called *continual common knowledge* and give a two-step method for transforming any solution to this problem into a round-optimal solution (optimal in the sense that no other solution outperforms it in all operating environments).[3] Since they study this problem in two of the benign failure models considered in our paper, it is again interesting to consider consistent formulations of this problem. Using a definition of continual common knowledge

[3]This definition of *optimal* is different from that given this paper; our optimal solutions must outperform every other solution in all operating environments.

strengthened in a way similar to the way we have defined strong common knowledge, Neiger and Bazzi [18] have generalized the work of Halpern, Moses, and Waarts to a general class of nonsimultaneous choice problems requiring consistent coordination. In particular, they have generalized the transformation and so can transform any solution to any consistent coordination problem into an optimal solution. The number of steps in this transformation depends on the number of actions from which the processors must choose. They also characterize problems for which there are optimal solutions (in the sense of this paper). Finally, they consider coordination problems requiring a particular kind of *termination* and show that solutions to such problems require a combination of eventual and continual common knowledge that they call *extended common knowledge.*

There still remain a few open problems to consider. First, if P≠NP, the precise complexity of solutions to consistent simultaneous choice problems in the general omissions model in which $n > 2t$ is still an open question: all we have established is that it is somewhere between NP and PSPACE. As noted by Moses and Tuttle, the precise complexity of solutions to nonconsistent simultaneous choice problems in this model is also an open problem. Furthermore, the complexity of solutions to nonconsistent problems is still open even when $n \leq 2t$, although we have established that solutions to *consistent* problems are impossible. Finally, we conjecture that the impossibility result of Section 6 applies to the nonsimultaneous *consistent coordination* problems mentioned above.

Acknowledgements

We thank Yoram Moses for enlightening discussions on the relationship between this work and that of Moses and Tuttle and for pointing out a minor error in an earlier version of $\mathcal{F}'_{MT}$. We thank Vassos Hadzilacos for his careful reading and constructive comments, and we thank the anonymous referees for their help.

References

[1] James E. Burns and Nancy A. Lynch. The Byzantine firing squad problem. *Advances*

in Computing Research: Parallel and Distributed Computing, 4:147–161, 1987. Also appears as Technical Report 275, MIT Laboratory for Computer Science.

[2] Brian A. Coan. A communication-efficient canonical form for fault-tolerant distributed protocols. In *Proceedings of the Fifth ACM Symposium on Principles of Distributed Computing*, pages 63–72, August 1986. A revised version appears in Coan's Ph.D. dissertation [3].

[3] Brian A. Coan. *Achieving Consensus in Fault-Tolerant Distributed Computer Systems: Protocols, Lower Bounds, and Simulations.* Ph.D. dissertation, Massachusetts Institute of Technology, June 1987.

[4] Brian A. Coan, Danny Dolev, Cynthia Dwork, and Larry Stockmeyer. The distributed firing squad problem. *SIAM Journal on Computing*, 18(5):990–1012, October 1989.

[5] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, October 1990.

[6] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: Crash failures. *Information and Computation*, 88(2):156–186, October 1990.

[7] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14:183–186, 1982.

[8] Ajei Gopal and Sam Toueg. Reliable broadcast in synchronous and asynchronous environments (preliminary version). In J.-C. Bermond and M. Raynal, editors, *Proceedings of the Third International Workshop on Distributed Algorithms*, volume 392 of *Lecture Notes on Computer Science*, pages 110–123. Springer-Verlag, September 1989.

[9] James Gray. Notes on database operating systems. In R. Bayer, R. M. Graham, and G. Seegmuller, editors, *Operating Systems: An Advanced Course*, volume 66 of *Lecture Notes on Computer Science.* Springer-Verlag, 1978. Also appears as Technical Report RJ2188, IBM Research Laboratory.

[10] Vassos Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations.* Ph.D. dissertation, Harvard University, June 1984. Technical Report 11-84, Department of Computer Science.

[11] Joseph Y. Halpern and Yoram Moses. A guide to the modal logic of knowledge and belief. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 480–490. Morgan-Kaufmann, August 1985.

[12] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, July 1990.

[13] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. A characterization of eventual Byzantine agreement. In *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, pages 333–346, August 1990.

[14] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[15] Butler Lampson and Howard Sturgis. Crash recovery in a distributed data storage system. Technical report, Computer Science Laboratory, Xerox, Palo Alto Research Center, Palo Alto, CA, 1976.

[16] C. Mohan, R. Strong, and S. Finkelstein. Methods for distributed transaction commit and recovery using Byzantine agreement within clusters of processors. In *Proceedings of the Second ACM Symposium on Principles of Distributed Computing*, pages 89–103, August 1983.

[17] Yoram Moses and Mark R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[18] Gil Neiger and Rida Bazzi. Using knowledge to optimally achieve coordination in distributed systems. In Yoram Moses, editor, *Proceedings of the Fourth Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 43–59. Morgan-Kaufmann, March 1992.

[19] Gil Neiger and Sam Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *Journal of Algorithms*, 11(3):374–419, September 1990.

[20] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[21] Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482, March 1986.

[22] Michael O. Rabin. Efficient solutions to the distributed firing squad problem. Private communication.

[23] Richard D. Schlichting and Fred B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, August 1983.