Programming Simultaneous Actions Using Common Knowledge[*]

Yoram Moses

Department of Applied Mathematics
Weizmann Institute
Rehovot, 76100
Israel

Mark R. Tuttle

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139
USA

Abstract: This work applies the theory of knowledge in distributed systems to the design of efficient fault-tolerant protocols. We define a large class of problems requiring coordinated, simultaneous action in synchronous systems, and give a method of transforming specifications of such problems into protocols that are *optimal in all runs*: for every possible input to the system and faulty processor behavior, these protocols are guaranteed to perform the simultaneous actions as soon as any other protocol could possibly perform them. This transformation is performed in two steps. In the first step, we extract directly from the problem specification a high-level protocol programmed using explicit tests for common knowledge. In the second step, we carefully analyze when facts become common knowledge, thereby providing a method of efficiently implementing these protocols in many variants of the omissions failure model. In the generalized omissions model, however, our analysis shows that testing for common knowledge is NP-hard. Given the close correspondence between common knowledge and simultaneous actions, we are able to show that no optimal protocol for any such problem can be computationally efficient in this model. The analysis in this paper exposes many subtle differences between the failure models, including the precise point at which this gap in complexity occurs.

May 19, 1987

Keywords: common knowledge, simultaneous action, Byzantine agreement, distributed firing squad, omissions failure model.

[*]This paper appeared in *Algorithmica*, 3(1):121–169, 1988. A preliminary version of this paper appeared in the *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, Toronto, October, 1986.

1 Introduction

The problem of ensuring proper coordination between processors in distributed systems whose components are unreliable is both important and difficult. There are generally two aspects to such coordination: the actions the different processors perform, and the relative timing of these actions. Both aspects are crucial, for instance, in maintaining consistent views of a distributed database. In particular, it is often most desirable to perform coordinated actions *simultaneously* at different sites of a system. It is therefore of great interest to study the design of protocols involving simultaneous actions, actions performed simultaneously by all processors whenever they are performed at all.

Dwork and Moses study in [DM] the design of protocols for simultaneous Byzantine agreement in the crash failure model. Their analysis focuses on determining necessary and sufficient conditions for reaching simultaneous Byzantine agreement in terms of the processors' states of knowledge about the system. As a result of this analysis, they derive a protocol for simultaneous Byzantine agreement with the unique property of being *optimal in all runs*; that is, their protocol halts as early as any protocol for the problem could, given the pattern of faulty processor behavior that occurs. In contrast, previous protocols do not adapt their behavior on the basis of faulty processor behavior, and hence always perform as poorly as they do in their *worst case* run. Implicit in the work of Dwork and Moses is a general method for obtaining optimal protocols for many problems involving simultaneous actions in the crash failure model. Their technical analysis, however, makes strong use of particular properties of the crash failure model, and does not extend to more complicated failure models.

This paper presents a novel approach to the design of fault-tolerant protocols in several variants of the more complex omissions failure model. We explicitly define a large class of *simultaneous choice problems*, a class intended to capture the essence of simultaneous coordination in synchronous systems. Many well-known problems, including simultaneous Byzantine agreement, distributed firing squad, etc., can be formulated as simultaneous choice problems. As the result of a delicate knowledge-based analysis in these failure models, we derive *at once* protocols that are *optimal in all runs* for all simultaneous choice problems: Each protocol is guaranteed to perform the desired simultaneous actions as soon as any protocol for the problem could, given the input to the system and the pattern of faulty processor behavior. (We will use *optimal* as shorthand for optimal in all runs.) Thus, we show how a knowledge-based analysis can be used as a general tool for the design of protocols for an entire class of problems. Our analysis applies to the crash failure model as well, and formally extends [DM] to the whole class of simultaneous choice problems.

Our approach is based on the close relationship between knowledge, communication, and action in distributed systems: A number of recent works ([HM], [DM], [Mo]) show that simultaneous actions are closely related to common knowledge. Informally, a fact is *common knowledge* if it is true, everyone knows it, everyone knows that everyone knows it, and so on ad infinitum. Notice that every processor performing a simultaneous action knows the action is being performed. In addition, since such actions are performed simultaneously by all processors, every processor knows that all processors know the action is being performed. This argument can be formalized and extended to show that when a simultaneous action is performed, it is common knowledge that the action is being performed. Consequently, a necessary condition for performing simultaneous actions is attaining common knowledge of particular facts. Interestingly, our work shows that in a precise sense this is also a sufficient condition: The problem of performing simultaneous actions reduces to the problem of attaining common knowledge of particular facts.

In deriving optimal protocols for simultaneous choice problems, we make explicit and direct use of the relationship between common knowledge and simultaneous actions. The derivation proceeds in two stages. In the first stage, we program the optimal protocols in a high-level language where processors' actions depend on explicit tests for common knowledge of certain facts. These high-level protocols are extracted directly from the problem specifications via a few simple manipulations. The second stage deals with effectively implementing these tests for common knowledge. We give a direct implementation of such tests in all variants of the omissions failure model we consider. As a result, our high-level protocols have effective implementations in these failure models as low-level, standard protocols that are optimal in all runs.

Consider, for example, the following version of the *distributed firing squad* problem (cf. [BL], [CDDS], [R]): An external source may send "start" signals to some of the processors in the system at unpredictable, possibly different, times. It is required that (i) if any nonfaulty processor receives a "start" signal, then all nonfaulty processors perform an irreversible "firing" action at some later point, (ii) whenever any nonfaulty processor "fires," all nonfaulty processors do so simultaneously, and (iii) if no processor receives a "start" signal, then no nonfaulty processor "fires." The high-level protocol we derive for this problem in the omissions model requires all processors to act as follows:

```
repeat every round
    send current view to every processor
until it is common knowledge that
                                    some processor received a "start" signal;
"fire" and halt.
```

Since we exhibit an effective implementation of the test for common knowledge embedded in this protocol, this high-level protocol can be transformed into a standard protocol that is optimal in all runs. No previous protocol for this problem suggested in the literature is optimal in all runs. Furthermore, in many cases this protocol "fires" much earlier than any other known protocol for this problem: In some cases, this protocol "fires" as soon as one round after the first "start" signal is received.

We show that optimal protocols for simultaneous choice problems can always be implemented in a communication efficient way, in all variants of the omissions model we consider. However, our direct implementation of tests for common knowledge is not computationally efficient: It requires processors to perform exponential time computations between consecutive rounds of communication. One of the major technical contributions of this paper is a method of efficiently implementing tests for common knowledge in several variants of the omissions failure model. In the standard omissions model, we provide a clean and concise method of efficiently implementing tests for common knowledge. The analysis underlying this method reveals the basic combinatorial structure underlying the omissions model, as well as crisply characterizing the set of facts that can be common knowledge at any point in the execution of a protocol. In the *receiving omissions* model, in which faulty processors may fail to receive messages rather than to send messages, testing for common knowledge is shown to be trivial. This exposes a significant difference between two seemingly symmetric failure models.

We are *not* able to efficiently implement tests for common knowledge in the *generalized omissions* model, in which faulty processors may fail both to send *and* to receive messages. In fact, we show that testing for common knowledge in this model in NP-hard. As a result, using the close relationship between common knowledge and simultaneous actions, we are able to show that no optimal protocol for any reasonable simultaneous choice problem can be computationally efficient unless P=NP. In particular, in this model there can be no computationally-efficient optimal protocol for the distributed firing squad problem stated above, for simultaneously performing Byzantine agreement (see [PSL], [DM]), and for most any other simultaneous problem. We consider another variant of the omissions model, called *generalized omissions with information*, in which it is assumed that the intended receiver of an undelivered message can test (and therefore knows) whether it or the sender is at fault. We show that the techniques used in the standard omissions model extend to this model as well, yielding computationally-efficient optimal protocols. As a result, we see that optimal protocols for simultaneous choice problems are computationally intractable in the generalized omissions model precisely because of the fact that in this model undelivered messages do not uniquely determine the set of faulty processors.

Thus, we show how to derive efficient optimal protocols in the omissions model, and we show that optimal protocols are intractable in the generalized omissions model. Since it is unrealistic to expect conventional processors (limited to polynomial-time computation) to follow such intractable protocols, it becomes becomes interesting to ask how well resource-bounded processors can perform simultaneous actions in the generalized omissions model. Analyzing this problem will require extending the theory of knowledge in distributed systems to account for the restricted computational power of such processors. Such an extension should give rise to notions of resource-bounded knowledge and common knowledge that closely correspond to the ability of resource-bounded processors to perform simultaneous actions. The need for a theory of resource-bounded knowledge has already been demonstrated, primarily by problems in which computational complexity is introduced by restricting the computational power of the adversary, thus allowing solutions involving encryption. This work, however, provides a more compelling indication of the need for such a theory, even for the analysis of simple problems in distributed computation that do not make such assumptions about the adversary.

Since some of the proofs in this paper are quite technical, their details may make it difficult to obtain a high-level understanding of this work. We strongly recommend that the reader skip all proofs on the first reading. The paper is organized as follows: Section 2 defines the model of distributed systems used in the paper, and Section 3 contains precise definitions of notions of knowledge in such a system. In Section 4 we define the notion of a *simultaneous choice problem*, a large class of problems involving coordinated simultaneous actions. Section 5 presents a uniform method of deriving an optimal high-level protocol from the specification of a simultaneous choice problem, using explicit tests for common knowledge. Section 6 deals with the problem of efficiently implementing tests for common knowledge of facts relevant to simultaneous choice problems. The analysis in Section 6 reveals interesting properties of the different failure models, and exposes fine distinctions between them. Finally, Section 7 contains some concluding remarks.

2 Model of a System

This section introduces a model of the distributed systems with which this paper is concerned. Our treatment extends and is closely related to that of [DM].

We consider synchronous systems of unreliable processors. Such a system consists of a finite collection $P = \{p_1, \ldots, p_n\}$ of n processors ($n \geq 2$), each pair of which is connected by a two-way communication link. Processors share a discrete global clock[1] that starts at time 0 and advances in increments of one. Communication in the system proceeds in a sequence of *rounds*, with round k taking place between time $k-1$ and time k. Between rounds of communication a processor may perform local computation and other internal actions. A processor starts in some *initial state* at time 0. Then, in every following round, the processor first sends a set of messages to other processors, and then receives messages sent to it by other processors during the same round. In addition, a processor may also receive requests for service from *clients* external to the system (think, for example, of a distributed airline reservation system). Actions resulting from the servicing of such requests may take a variety of forms, including the initiation of various activities within the system by sending certain messages to other processors in later rounds. Each message is assumed to be tagged with the identities of the sender and intended receiver of the message, as well as the round in which it is sent; similarly for each request. At any given time, a processor's *message history* consists of the list of messages it has received from the other processors, and a processor's *input history* consists of its initial state together with

[1] We assume the existence of a shared global clock for ease of exposition. The analysis performed in this paper applies even in synchronous systems in which processors have local clocks and start operating in an arbitrarily staggered order.

the requests it has received from the system's external clients. A processor's *view* at any given time consists of its message history, its input history, the time on the global clock, and the processor's identity. For technical reasons, it will be convenient to talk about processors' views at negative times (before time 0). A processor's view at a negative time is defined to be a distinguished *empty* view.

We think of the processors as following a *protocol*, which specifies exactly what messages each processor is required to send during a round (and what other actions the processor is required to perform) as a deterministic function of the processor's view. While a protocol determines the behavior of each processor (as a function of its view), processors are unreliable and some of them may be *faulty*, the rest being *nonfaulty*. Both faulty and nonfaulty processors faithfully follow the protocol, their behaviors differing only in the messages they successfully send and receive.[2] A nonfaulty processor sends every message it is required by the protocol to send, and receives every message sent to it by other processors, in all rounds of communication. A faulty processor, however, may fail to send or receive certain messages. We will consider a number of different processor *failure models*: (i) the *omissions* model ([MSF]), in which a faulty processor receives every message sent to it, but sends only an arbitrary (not necessarily strict) subset of the messages it is required to send; (ii) the *receiving omissions* model, in which a faulty processor sends every message it is required to send, but receives only an arbitrary subset of the messages sent to it; (iii) the *generalized omissions* model, in which a faulty processor may both send only an arbitrary subset of the messages it is required to send *and* receive only an arbitrary subset of the messages sent to it; and (iv) *generalized omissions with information*, which differs from the generalized omissions model in that a processor not receiving a message from another processor can determine whether it or the sender is at fault.

An infinite execution of a protocol in a system is called a *run* of the protocol. We identify a run with the complete history of events that take place during the run, from time 0 until the end of time. This includes each processor's complete input history, complete message history, and, if the processor is faulty in the run, a description of its behavior during each round (formalized in the following paragraph). A pair (ρ, ℓ), where ρ is a run and ℓ is a natural number, is called a *point*, and represents the state of the system after the first ℓ rounds of ρ. We denote processor q's view at the point (ρ, ℓ) by $v(q, \rho, \ell)$.

We now define the notion of a failure pattern, a formal description of faulty processor behavior during a run. The notion of a failure pattern in each variant of the omissions model is a suitable restriction of the general definition given here. Remember that a faulty processor may fail to send or receive certain messages. It is therefore natural to define the *faulty behavior* of a processor p to be a pair of functions S and R mapping

[2] Intuitively, processors attempt to send and receive all required messages. Failures are caused by faulty input/output ports. However, we will often speak of processors failing to send or receive a given message when we mean that the message was not successfully sent or received, respectively.

round numbers to sets of processors. Intuitively, these are the processors to which p fails to send and receive messages, respectively, during each round. A *failure pattern* is a collection of faulty behaviors $\langle S_i, R_i\rangle$, one for each processor p_i. The processor p_i is said to be *faulty* in such a failure pattern if either of the sets $S_i(k)$ or $R_i(k)$ is nonempty for some k, in which case p_i is said to *fail* during round k. The *failure pattern* of a run is a failure pattern with the property that in every round k each processor p_i sends no messages to processors in $S_i(k)$ but sends all required messages to processors not in $S_i(k)$, and receives no messages from processors in $R_i(k)$ but receives all messages sent to it by processors not in $R_i(k)$. Given a run ρ, if γ_i is the complete input history of processor p_i in ρ, then we say that $\gamma = (\gamma_1, \ldots, \gamma_n)$ is the *input* to ρ. A pair (π, γ), where π is a failure pattern and γ is an input, is called an *operating environment*. Notice that a run is uniquely determined by a protocol and an operating environment. Two runs of two different protocols are said to be *corresponding runs* if they have the same operating environment. The fact that an operating environment is independent of the protocol will allow us to compare different protocols according to their behavior in corresponding runs.

In this work, we study the behavior of protocols in the presence of a bounded number of failures (of a particular type) and a given setting of possible inputs. It is therefore natural to identify a *system* with the set of all possible runs of a given protocol under such circumstances. Formally, a system is identified with the set of runs of a protocol $\mathcal{P}$ by $n \geq 2$ processors, at most $t \leq n-2$ of which may be faulty (in the sense of a particular failure model $\mathcal{M}$), where the complete input history of each processor p_i is an element of a set Γ_i. We denote this set of runs by the tuple $\Sigma = (n, t, \mathcal{P}, \mathcal{M}, \Gamma_1, \ldots, \Gamma_n)$. Our definition of a system ensures that the input to the system is orthogonal to, and hence carries no information about, the failure pattern. In addition, since the set of possible inputs in the system has the form $\Gamma_1 \times \cdots \times \Gamma_n$, one processor's input contains no information about any other processor's input, and hence the only way in which processors obtain information about other processors' input is via messages communicated between the processors in the system.

While a protocol may be thought of as a function of processors' views, protocols for distributed systems (as well as protocols for sequential and parallel computation) are typically written for systems of arbitrarily large size. In this sense, the actions and messages required of a processor by a protocol actually depend on the number of processors in the system (and perhaps the bound on the number of failures) as well as the view of the processor. Therefore, we formally define a *protocol* to be a function from n, t, and a processor's view to a list of actions the processor is required to perform, followed by a list of messages the processor is required to send in the following round.[3] Since each protocol is defined for systems of arbitrary size, it is natural to define a

[3] Notice that processors must compute this function by following some algorithm. Thus, while we formally define a protocol to be a function, it is convenient to maintain both views of a protocol as a function and an algorithm.

class of systems to be a collection of systems $\{\Sigma(n,t) : n \geq t+2 \geq 0\}$, where $\Sigma(n,t) = (n, t, \mathcal{P}, \mathcal{M}, \Gamma_1, \ldots, \Gamma_n)$ for some fixed protocol $\mathcal{P}$, failure model $\mathcal{M}$, and input sets Γ_i.

3 Definition of Knowledge

Our analysis makes essential use of reasoning about processors' knowledge at various points in the execution of a protocol. This section contains precise definitions of the notions of knowledge we use. For the purpose of these definitions, we assume that a particular system, a set of runs as defined in the previous section, is fixed ahead of time. All runs mentioned will be runs of this system, and all points will be points in such runs. Our treatment is a modification of that of [DM] and [HM].

We assume the existence of an underlying logical language for representing all relevant *ground* facts — facts about the system that do not explicitly mention processors' knowledge (for example, *"the value of register x is 0"*, or *"processor p_i failed in round 3"*). Formally, a ground fact φ will be identified with a set of points $\tau(\varphi)$. Intuitively, this is the set of points at which the fact holds. A ground fact φ is said to *hold* at a point (ρ, ℓ), denoted $(\rho, \ell) \models \varphi$, iff $(\rho, \ell) \in \tau(\varphi)$. We will define various ground facts as we go along. The set of points corresponding to these facts will be clear from context. A fact is said to be *valid* if it is true of all points in all systems. A fact is said to be *valid in the system* for a given system if it is true of all points in the system.

We now define what facts a processor is said to "know" at any given point (ρ, ℓ) in the system. Roughly speaking, a processor p_i is said to know a fact φ if φ is guaranteed to hold, given p_i's view of the run. More formally, we say p_i *knows* φ at (ρ, k), denoted $(\rho, k) \models K_i\varphi$, if $(\rho', k) \models \varphi$ for all points (ρ', k) satisfying $v(p_i, \rho, k) = v(p_i, \rho', k)$. This definition of knowledge is essentially the *total view* interpretation of [HM]. It is "external," in the sense that a processor is ascribed knowledge based solely on the processor's information, and not, say, on the local computation it performs or on its computational power. Notice that a processor's knowledge at a given point depends on the system as well as on the specific run. Thus, implicit in the definition of $(\rho, \ell) \models \varphi$ is the system relative to which knowledge is determined. Throughout the paper it will be clear from context what the relevant system should be whenever "$\models$" is used.

We will find it useful to extend this definition of knowledge to sets of processors as well. The *view* of a set of processors $G \subseteq P$ at (ρ, k), denoted $v(G, \rho, k)$, is defined by:

$$v(G, \rho, k) \quad \stackrel{\text{def}}{=} \quad \{v(p, \rho, k) : p \in G\}.$$

Thus, roughly speaking, G's view is simply the joint view of its members. We say that the group G *implicitly knows* φ at (ρ, k), denoted $(\rho, k) \models I_G\varphi$, if for all points (ρ', k) satisfying $v(G, \rho, k) = v(G, \rho', k)$ it is the case that $(\rho', k) \models \varphi$. In the particular case that G is the singleton set $\{p_i\}$, the notions of I_G and K_i coincide. Intuitively, G

implicitly knows φ if the joint view of G's members guarantees that φ holds. Notice that if processor p knows φ and processor q knows $\varphi \supset \psi$, then together they implicitly know ψ, even if neither of them knows ψ individually. The notion of implicit knowledge was first defined in [HM].

The notions of knowledge and implicit knowledge defined above are closely related to modal logics that have been extensively studied by philosophers (see [Hi]). We say that an operator M *has the properties of the modal system S5* if it satisfies a) if φ is valid in the system then $M\varphi$ is valid in the system; and the following formulas are valid: b) $M\varphi \supset \varphi$; c) $(M\varphi \wedge M(\varphi \supset \psi)) \supset M\psi$; d) $M\varphi \supset MM\varphi$; and e) $\neg M\varphi \supset M\neg M\varphi$. The definitions of knowledge and implicit knowledge given above immediately imply the following (cf. [HM2], [DM]):

Proposition 1: The operators K_i and I_G have the properties of S5.

Finally, the state of common knowledge among a group of processors will be central to our analysis. Its central role will result from the close correspondence between common knowledge among the members of a group and simultaneous actions performed by the group. Roughly speaking, as we mentioned in the introduction, a fact φ is common knowledge to a given group if φ holds, everyone in the group knows φ, everyone knows that everyone knows φ, and so on ad infinitum. Formally defining common knowledge, however, must be done with great care. The problem is that the groups of interest are not always explicitly given as fixed subsets of P. For example, we will be most interested in facts that are common knowledge to the group $\mathcal{N}$ of nonfaulty processors. In any given context (in this case, any given run), this group is a fixed set of processors. But the precise identity of $\mathcal{N}$ varies from one context to another. This motivates us to define common knowledge to a slightly more general notion of groups of processors: An *indexical set* $\mathcal{S}$ of processors is a function mapping points to sets of processors. That is, $\mathcal{S} : (\rho, \ell) \mapsto \mathcal{S}(\rho, \ell)$, where $\mathcal{S}(\rho, \ell) \subseteq P$. The notion of an indexical set is a direct generalization of the notion of a fixed set of processors. In particular, we can identify a fixed set of processors with a constant indexical set. The group $\mathcal{N}$ of nonfaulty processors, the group P of all processors, the group of all processors that haven't displayed faulty behavior by the current time, and many other groups of interest are all indexical sets of processors.

The first step in defining what it means for a fact to be common knowledge among the members of a given group of processors is to determine what it means for everyone in the group to know a fact. For a fixed set G, "everyone in G knows φ," denoted $E_G\varphi$, is customarily defined by $E_G\varphi \equiv \bigwedge_{p_i \in G} K_i\varphi$ (see [HM]). In extending this notion to indexical sets, however, a subtle decision must be made. The immediate generalization of this definition is to define $E_\mathcal{S}\varphi \equiv \bigwedge_{p_i \in \mathcal{S}} K_i\varphi$. This generalization, however, does not yield a notion of common knowledge that closely corresponds to $\mathcal{S}$'s ability to perform simultaneous actions (see Lemma 4 below). Given that G is a fixed set, and that the knowledge operator K_i satisfies property (a) of S5 given above, it follows that $p_i \in G \equiv K_i(p_i \in G)$ is valid. Therefore, an equivalent definition of $E_G\varphi$ is $E_G\varphi \equiv \bigwedge_{p_i \in G} K_i(p_i \in G \ \supset \varphi)$. We choose this form of "everyone knows" as the appropriate generalization to indexical sets. Formally, given an indexical set $\mathcal{S}$, we define $E_{\mathcal{S}}\varphi$, essentially corresponding to *everyone in $\mathcal{S}$ knows* φ, by

$$E_{\mathcal{S}}\varphi \stackrel{\text{def}}{=} \bigwedge_{p_i \in \mathcal{S}} K_i(p_i \in \mathcal{S} \ \supset \varphi).$$

Thus, $E_{\mathcal{S}}\varphi$ holds exactly if every member of $\mathcal{S}$ knows that, if it is a member of $\mathcal{S}$, then φ holds.

We now define φ *is common knowledge to* $\mathcal{S}$, denoted $C_{\mathcal{S}}\varphi$, by

$$C_{\mathcal{S}}\varphi \stackrel{\text{def}}{=} \varphi \wedge E_{\mathcal{S}}\varphi \wedge E_{\mathcal{S}}E_{\mathcal{S}}\varphi \wedge \cdots \wedge E_{\mathcal{S}}^m\varphi \wedge \cdots.$$

In other words, $(\rho, \ell) \models C_{\mathcal{S}}\varphi$ iff both $(\rho, \ell) \models \varphi$ and for all $m \geq 1$ it is the case that $(\rho, \ell) \models E_{\mathcal{S}}^m\varphi$. Thus, roughly speaking, a fact is common knowledge if it is true, everyone knows it, everyone knows that everyone knows it, etc. The definitions of $E_{\mathcal{S}}$ and $C_{\mathcal{S}}$ directly generalize the standard notions from [HM] and [DM].

A useful tool for thinking about $E_{\mathcal{S}}^m\varphi$ and $C_{\mathcal{S}}\varphi$ is an undirected graph whose nodes are the points of the system, in which two points (ρ, k) and (ρ', k) are connected by an edge iff some processor p that is a member of both $\mathcal{S}(\rho, k)$ and $\mathcal{S}(\rho', k)$ has the same view at both (ρ, k) and (ρ', k). This graph is called the *similarity graph relative to* $\mathcal{S}$. For example, if $\mathcal{S}$ is the set $\mathcal{N}$ of nonfaulty processors, two points are connected by an edge in the similarity graph iff there is a processor that is nonfaulty at both points, and has the same view at both points. An easy argument by induction on m shows that $(\rho, k) \models E_{\mathcal{S}}^m\varphi$ iff $(\rho', k) \models \varphi$ for all points (ρ', k) of distance at most m from (ρ, k) in this graph. It follows that $(\rho, k) \models C_{\mathcal{N}}\varphi$ iff $(\rho', k) \models \varphi$ for all points (ρ', k) in the connected component of (ρ, k). Two points (ρ, ℓ) and (ρ', ℓ) are said to be *similar relative to* $\mathcal{S}$, denoted $(\rho, \ell) \overset{\mathcal{S}}{\sim} (\rho', \ell)$, if they are in the same connected component of the similarity graph relative to $\mathcal{S}$. Since the indexical set $\mathcal{S}$ is generally clear from context (most often being the set $\mathcal{N}$ of nonfaulty processors), we denote similarity by $\sim$ without the superscript $\mathcal{S}$. We thus have:

Theorem 2: $(\rho, k) \models C_{\mathcal{S}}\varphi$ iff $(\rho', k) \models \varphi$ for all points (ρ', k) satisfying $(\rho, k) \sim (\rho', k)$.

Our analysis will exploit this relationship between common knowledge and the similarity graph. The similarity graph will provide us with a useful combinatorial tool with which to study when facts become common knowledge.

One of the useful properties of common knowledge is the so-called "fixpoint axiom" (see [HM])

$$C_{\mathcal{S}}\varphi \equiv E_{\mathcal{S}}C_{\mathcal{S}}\varphi,$$

which states that common knowledge is a fixpoint of the E_S operator (provided $\mathcal{S}$ is nonempty, as will invariably be the case in this work). It implies that a fact's being common knowledge is in a sense "public:" a fact can be common knowledge to a group of processors only if all members of the group know that it is common knowledge. This axiom also implies that when a fact becomes common knowledge, it becomes common knowledge to all relevant processors simultaneously. Another useful fact about common knowledge is captured by the following *induction rule*:

If	$\varphi \supset E_S\varphi$	is valid in the system,
then	$\varphi \supset C_S\varphi$	is valid in the system.

Roughly speaking, the induction rule implies that if a fact is "public" to a group of processors, in the sense that whenever it holds it is known to all members of the group, then whenever it holds it is in fact common knowledge. These are two essential properties of common knowledge that will prove useful to our analysis. In addition, we can also show the following:

Proposition 3: The operator C_S has the properties of S5.

According to our definitions, facts about the system are properties of points: they are either true or false at any given point. It is often useful to be able to refer to facts as being about things other than points (e.g., properties of runs). In general, a fact φ is said to be a *fact about* X if fixing X determines the truth (or falsity) of φ. For example, a fact φ is said to be a fact *about the input* if fixing the input determines whether or not φ holds. That is, for any given input γ, either φ holds at all points (ρ, k) where ρ is a run with input γ, or φ holds at no such point. The meaning of a fact being *about the operating environment*, *about the existence of failures*, *about the first k rounds*, etc., are similarly defined.

4 Simultaneous Choice Problems

In order to study in a uniform and general way the design of protocols for problems involving coordinated simultaneous action, a definition of this class of problems is required. Lacking a most general definition, we focus on the class of *simultaneous choice problems*, a large class of problems that capture the essence of such coordinated action in a distributed environment. Roughly speaking, these problems require that one of a number of alternative actions be performed (or "chosen") simultaneously by the nonfaulty processors, where for each action we are given conditions under which the action *must* be performed and conditions under which its performance is forbidden. In addition to these conditions, the specification of such a problem must also determine the possible operating environments in which such a choice is to be made, by specifying what inputs each processor may possibly receive and what types of processor failures are possible.

Formally, a *simultaneous action* a is an action having two associated conditions $pro(a)$ and $con(a)$, both facts about the operating environment. A *simultaneous choice problem* $\mathcal{C}$ is a problem determined by a set $\{a_1, \ldots, a_m\}$ of simultaneous actions and their associated conditions, together with a failure model $\mathcal{M}$, and a set Γ_j of complete input histories for each processor p_j. Intuitively, we will require that every run ρ of a protocol implementing $\mathcal{C}$ satisfy the following conditions:

(i) each nonfaulty processor performs at most one of the a_i's,

(ii) any a_i performed by some nonfaulty[4] processor is performed *simultaneously* by all of them,

(iii) a_i is performed by all nonfaulty processors if ρ satisfies $pro(a_i)$, and

(iv) a_i is *not* performed by any nonfaulty processor if ρ satisfies $con(a_i)$.

More formally, a protocol $\mathcal{P}$ and the simultaneous choice $\mathcal{C}$ *determine* a class of systems $\{\Sigma(n,t) : n \geq t+2\}$, where $\Sigma(n,t) = (n, t, \mathcal{P}, \mathcal{M}, \Gamma_1, \ldots, \Gamma_n)$. We say that $\mathcal{P}$ *implements* $\mathcal{C}$ if every run of every system in the class determined by $\mathcal{P}$ and $\mathcal{C}$ satisfies the conditions (i)-(iv) above. A simultaneous choice problem is said to be *implementable* (or *satisfiable*) if there is a protocol that implements it.

In addition to simultaneous choice problems, we also consider the closely related class of *strict* simultaneous choice problems. Both classes are specified in essentially the same way, except that runs of a protocol implementing a *strict* simultaneous choice are required to satisfy the modified condition

(i′) each nonfaulty processor performs *exactly* one of the a_i's,

together with conditions (ii)-(iv) above. All of the results in this paper hold for strict simultaneous choice problems as well as simultaneous choice problems, and henceforth we will typically mention only simultaneous choice problems explicitly.

Having formally defined simultaneous choice problems (and *strict* simultaneous choice problems), let us consider when the specification of such a problem disallows performing a simultaneous action a_i. Clearly, if $con(a_i)$ holds then performing a_i is disallowed. In addition, since by condition (i) no more than one action may be performed by the nonfaulty processors in any given run, the condition $pro(a_j)$, for some $j \neq i$, requires a_j to be

[4] We have chosen the set $\mathcal{N}$ of nonfaulty processors as the set of processors required to perform actions simultaneously, but the notion of a simultaneous choice problem may be stated in terms of many other similar (indexical) sets of processors, including the set P of all processors, with the analysis in this section and the next one carrying through without change.

performed, and hence also disallows a_i. It is easy to see that these are the only conditions under which performing a_i is disallowed. This motivates the following definition:

$$\mathit{enabled}(a_i) \stackrel{\mathrm{def}}{=} \neg \mathit{con}(a_i) \wedge \bigwedge_{j \neq i} \neg \mathit{pro}(a_j).$$

Our discussion above implies that the performance of an action a_i is allowed by the problem specification iff the condition $\mathit{enabled}(a_i)$ is satisfied. Notice that it is possible for several of the conditions $\mathit{enabled}(a_i)$ to hold at once, in which case performance of any of the enabled actions is allowed by the problem specification. In addition, it is easy to see that the formulas $\mathit{con}(a_i) \supset \neg \mathit{enabled}(a_i)$ and $\mathit{pro}(a_i) \supset \neg \mathit{enabled}(a_j)$ $(j \neq i)$ are valid in any system in which processors follow a protocol implementing a simultaneous choice. Finally, notice that because the conditions $\mathit{pro}(a_j)$ and $\mathit{con}(a_j)$ are facts about the operating environment, so is each condition $\mathit{enabled}(a_i)$.

The definition of a simultaneous choice problem is fairly abstract. However, many familiar problems requiring simultaneous action by a group of processors are instances of a simultaneous choice or strict simultaneous choice. In all known cases, the conditions $\mathit{pro}(a_i)$ and $\mathit{con}(a_i)$ are facts about the input and the existence of failures. (By *the existence of failures* we mean whether *any failure whatsoever* occurs during the run. Some problems allow the nonfaulty processors to display default behavior in the presence of failures; see [LF].) For example, the distributed firing squad problem is a simultaneous choice consisting of a single "firing" action a, with the condition $\mathit{pro}(a)$ being the receipt of a "start" signal by a nonfaulty processor, and the condition $\mathit{con}(a)$ being that no processor receives a "start" signal. The condition $\mathit{enabled}(a)$ is simply that *some* processor receives a "start" signal. Each set Γ_j of possible inputs simply allows for a "start" message to be delivered to any processor at any time. The simultaneous Byzantine agreement problem (see [DM], [PSL]) is an example of a *strict* simultaneous choice. This problem consists of an action a_0 of "deciding 0" and an action a_1 of "deciding 1." Each set Γ_j of possible inputs consists of two possible inputs: one starting with initial value 0 and receiving no further external input during the run, and the other starting with initial value 1. The condition $\mathit{pro}(a_0)$ is that all initial values are 0, and the condition $\mathit{pro}(a_1)$ is that all initial values are 1. The conditions $\mathit{con}(a_0)$ and $\mathit{con}(a_1)$ are both taken to be *false*. Here the condition $\mathit{enabled}(a_0)$ is that some initial value is 0, and the condition $\mathit{enabled}(a_1)$ is that some initial value is 1. Since for most assignments of initial values both $\mathit{enabled}(a_0)$ and $\mathit{enabled}(a_1)$ hold, it is typically the case that deciding either 0 or 1 is acceptable. Simultaneous Byzantine agreement is a *strict* simultaneous choice, since the processors are required to decide either 0 or 1 in every run. Other related problems that may also be formulated as (strict) simultaneous choice problems include weak Byzantine agreement and the Byzantine Generals problem (see [F]).

Having formally defined the notion of a simultaneous action, we are now in a position to carefully state the relationship between simultaneous actions and common knowledge mentioned in the introduction: When a simultaneous action is performed, it is common knowledge that the action is being performed. The statement we actually prove is that when such an action is performed, it is common knowledge that the action is enabled.

Lemma 4: Let ρ be a run of a protocol implementing a simultaneous choice $\mathcal{C}$. If the action a_i of $\mathcal{C}$ is performed by a nonfaulty processor at time ℓ in ρ, then $(\rho, \ell) \models C_{\mathcal{N}} \mathit{enabled}(a_i)$.

Proof: Let φ be the fact "a_i is being performed by a nonfaulty processor." A processor p_j performing the action a_i clearly knows that it is performing a_i. This processor therefore also knows that if it is nonfaulty then a_i is being performed by a nonfaulty processor. Since ρ is a run of a protocol implementing $\mathcal{C}$, the action a_i is performed simultaneously by all nonfaulty processors whenever it is performed by a single nonfaulty processor. It follows that whenever φ holds, so does $E_{\mathcal{N}}\varphi$, and hence $\varphi \supset E_{\mathcal{N}}\varphi$ is valid in the system. The induction rule implies that $\varphi \supset C_{\mathcal{N}}\varphi$ is valid in the system as well. Notice that $\varphi \supset \mathit{enabled}(a_i)$ is valid in the system. It thus follows that $C_{\mathcal{N}}\varphi \supset C_{\mathcal{N}} \mathit{enabled}(a_i)$ is valid in the system, and hence so is $\varphi \supset C_{\mathcal{N}} \mathit{enabled}(a_i)$. Thus, $(\rho, \ell) \models \varphi$ implies $(\rho, \ell) \models C_{\mathcal{N}} \mathit{enabled}(a_i)$, and we are done. □

In the above proof, the essential fact that $\varphi \supset E_{\mathcal{N}}\varphi$ is valid in the system depends crucially on our definition of $E_{\mathcal{N}}\varphi$. A processor p performing a_i knows that a_i is being performed, but since a nonfaulty processor might not know that it is nonfaulty, p might *not* know that a_i is being performed by a nonfaulty processor. The processor p *does* know, however, that if *it* (p itself) is nonfaulty, then a nonfaulty processor is performing a_i. It is for this reason that we have been led to choose our definition of $E_{\mathcal{N}}\varphi$ as we have, as discussed in the previous section.

5 Optimal Protocols

In this section, we show how to extract a high-level optimal protocol for a simultaneous choice problem directly from its specification. (As mentioned in the introduction, we use the word *optimal* as shorthand for optimal in all runs; recall that this optimality is in terms of the number of rounds required to perform a simultaneous choice.) We begin by considering a simple class of protocols that will serve as a building block in the design of such optimal protocols. Recall that a protocol is a function specifying the actions a processor should perform and the messages it should send as a function of n, t, and the processor's view. Thus, we can think of a protocol as having two components: an *action* component and a *message* component. A protocol is said to be a *full-information protocol* (cf. [Ha], [FL], [PSL]) if its message component is:

```
repeat every round
        send current view to all processors
forever.
```

Intuitively, since such a protocol requires that all processors send all of the information available to them in every round, one would expect this protocol to give each processor as much information about the operating environment as any protocol could. In particular, the following result shows that if a processor cannot distinguish two operating environments during runs of a full-information protocol, then the processor cannot distinguish these operating environments during runs of any other protocol.

Lemma 5: Let ρ and ρ' be runs of a full-information protocol $\mathcal{F}$, and let ζ and ζ' be runs of an arbitrary protocol $\mathcal{P}$ corresponding to ρ and ρ', respectively. For all processors q and times ℓ, if $v(q,\rho,\ell) = v(q,\rho',\ell)$ then $v(q,\zeta,\ell) = v(q,\zeta',\ell)$.

Proof: We proceed by induction on the time ℓ. The case of $\ell = 0$ is immediate since q must have the same initial state in both ρ and ρ', and hence also in ζ and ζ'. Suppose $\ell > 0$ and the inductive hypothesis holds for all processors p at time $\ell - 1$. The view of q at time ℓ is determined by its view at time $\ell - 1$, the (external) input it receives during round ℓ, and the messages it receives during round ℓ. Since q has the same view at time $\ell - 1$ in ρ and ρ', by the inductive hypothesis, the same is true in ζ and ζ'. Since q receives the same input during round ℓ in ρ and ρ', the same is true in ζ and ζ'. If q does not receive a message from p during round ℓ in ρ and ρ', then both operating environments determine that no message from p to q during round ℓ is delivered. Thus, q does not receive a message from p during round ℓ in either ζ or ζ'. If q *does* receive a message from p during round ℓ in ρ and ρ', then both operating environments determine that any message from p to q during round ℓ is delivered. If q receives a message from p during round ℓ of ρ and ρ', then since q must receive the same message from p in both ρ and ρ', the view of p must be the same at time $\ell - 1$ in ρ and ρ'. By the inductive hypothesis, p's view at time $\ell - 1$ must also be the same in ζ and ζ'. Since $\mathcal{P}$ is a deterministic function of processor views, q receives the same messages from p during round ℓ in ζ and ζ'. Thus, q has the same view at time ℓ in ζ and ζ'. □

Thus, roughly speaking, processors learn the most about the operating environment during runs of full-information protocols. The following corollary of Lemma 5 shows that facts about the operating environment become common knowledge during runs of such protocols at least as soon as they do during runs of any other protocol. This result captures in a precise sense a property of full-information protocols that is essential to our analysis.

Corollary 6: Let φ be a fact about the operating environment. Let ρ and ζ be corresponding runs of a full-information protocol $\mathcal{F}$ and an arbitrary protocol $\mathcal{P}$, respectively. If $(\zeta,\ell) \models C_{\mathcal{N}}\varphi$ then $(\rho,\ell) \models C_{\mathcal{N}}\varphi$.

Proof: Suppose that $(\zeta,\ell) \models C_{\mathcal{N}}\varphi$. We will prove that $(\rho,\ell) \models C_{\mathcal{N}}\varphi$ by showing that $(\rho',\ell) \models \varphi$ for all runs ρ' of $\mathcal{F}$ such that $(\rho,\ell) \sim (\rho',\ell)$. Fix ρ', and let ζ' be the run of $\mathcal{P}$

corresponding to ρ'. Lemma 5 and a simple inductive argument on the distance between (ρ,ℓ) and (ρ',ℓ) in the similarity graph show that $(\rho,\ell) \sim (\rho',\ell)$ implies $(\zeta,\ell) \sim (\zeta',\ell)$. Since $(\zeta,\ell) \models C_{\mathcal{N}}\varphi$, we have $(\zeta',\ell) \models \varphi$. Since corresponding runs satisfy the same facts about the operating environment, $(\zeta',\ell) \models \varphi$ implies $(\rho',\ell) \models \varphi$. It follows that $(\rho,\ell) \models C_{\mathcal{N}}\varphi$. □

We are now in a position to describe how to construct optimal protocols for simultaneous choice problems. Recall that when a simultaneous action a_i is performed, Lemma 4 implies that *enabled*(a_i) must be common knowledge. Since *enabled*(a_i) is a fact about the operating environment, Corollary 6 implies that *enabled*(a_i) becomes common knowledge in runs of a full-information protocol as soon at it does in corresponding runs of any other protocol. Thus, given an effective test that the nonfaulty processors can use to determine whether *enabled*(a_i) is common knowledge, a test we denote by *test-for-*$C_{\mathcal{N}}$*enabled*(a_i), the following protocol $\mathcal{F}_{\mathcal{C}}$ is an optimal protocol for $\mathcal{C}$:

```
no_action_performed ← true;
repeat every round
   if no_action_performed and test-for-$C_{\mathcal{N}}$enabled($a_i$) returns true for some $a_i$
      then
         $j$ ← min {$i$ : test-for-$C_{\mathcal{N}}$enabled($a_i$) returns true},
         perform $a_j$,
         no_action_performed ← false;
   send current view to every processor;
forever.
```

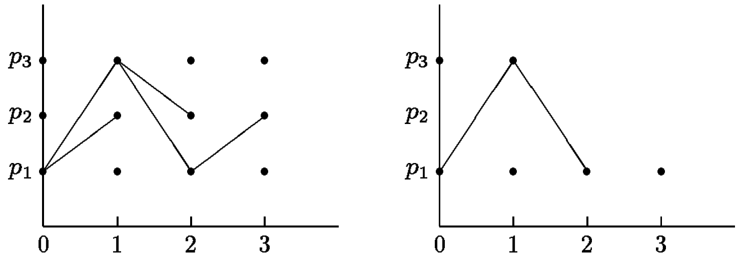
Before formally proving that $\mathcal{F}_{\mathcal{C}}$ is an optimal protocol, we must define more formally the tests for common knowledge that appear in $\mathcal{F}_{\mathcal{C}}$. Recall that the fixpoint axiom implies $C_{\mathcal{N}}\varphi \supset E_{\mathcal{N}}C_{\mathcal{N}}\varphi$ is valid. This guarantees that $C_{\mathcal{N}}\varphi$ follows from the view of each nonfaulty processor whenever $C_{\mathcal{N}}\varphi$ holds. Notice, that $C_{\mathcal{N}}\varphi$ is *not* guaranteed to follow from the view of a faulty processor. It is therefore natural to define a *test for common knowledge of* φ, denoted as above by *test-for-*$C_{\mathcal{N}}\varphi$, to be a test that, given the view of a nonfaulty processor at (ρ,ℓ) (together with n and t), returns **true** iff $C_{\mathcal{N}}\varphi$ holds at (ρ,ℓ). Such a test may return either **true** or **false** when given the view of a faulty processor. Let us denote by $\mathcal{A}_j(\rho,\ell)$ the set of actions a_i such that *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) returns **true** when given the view of p_j at (ρ,ℓ). Notice that if p_j is nonfaulty, then $\mathcal{A}_j(\rho,\ell)$ is precisely the set of actions a_i such that $C_{\mathcal{N}}$*enabled*(a_i) holds at (ρ,ℓ). It follows that for all nonfaulty processors p_j the sets $\mathcal{A}_j$ are equal at all times. In particular, all become nonempty at the same time (as soon as *enabled*(a_i) becomes common knowledge for some a_i). Thus, if all processors p_j choose the action of least index from $\mathcal{A}_j$ as soon as this set becomes nonempty, as required by $\mathcal{F}_{\mathcal{C}}$, then all nonfaulty processors choose the same action simultaneously. We can now prove that $\mathcal{F}_{\mathcal{C}}$ is an optimal protocol for $\mathcal{C}$. (Recall that a simultaneous choice problem is *implementable* iff there exists a protocol that implements it.)

Theorem 7: If $\mathcal{C}$ is an implementable simultaneous choice problem, then $\mathcal{F}_c$ is an optimal protocol for $\mathcal{C}$.

Proof: We first prove that nonfaulty processors perform actions in runs of $\mathcal{F}_c$ as soon as they do in corresponding runs of any protocol implementing $\mathcal{C}$. Let ρ be a run of $\mathcal{F}_c$, and let ζ be the corresponding run of a protocol implementing $\mathcal{C}$. Lemma 4 implies that if a_i is performed by a nonfaulty processor at time ℓ in ζ, then $(\zeta, \ell) \models C_{\mathcal{N}} \mathit{enabled}(a_i)$. Since $\mathit{enabled}(a_i)$ is a fact about the operating environment, Corollary 6 implies that $(\rho, \ell) \models C_{\mathcal{N}} \mathit{enabled}(a_i)$. As a result, $\mathcal{A}_j(\rho, \ell)$ must be nonempty for all nonfaulty processors p_j, and hence each must perform an action in ρ no later that time ℓ. It follows that nonfaulty processors perform actions in runs of $\mathcal{F}_c$ as soon as they do in corresponding runs of any protocol implementing $\mathcal{C}$. We now show that $\mathcal{F}_c$ actually implements $\mathcal{C}$. Let ρ be a run of $\mathcal{F}_c$. First, it is obvious from the definition of $\mathcal{F}_c$ that each nonfaulty processor performs at most one action in ρ. (If $\mathcal{C}$ is an implementable *strict* simultaneous choice, then the preceding discussion shows that the nonfaulty processors perform *exactly* one action in ρ.) Second, if a nonfaulty processor p_j performs an action a_i at time ℓ during ρ, then time ℓ is the first time at which $\mathcal{A}_j(\rho, k)$ is nonempty, and a_i is the action of least index in this set. Since $\mathcal{A}_j(\rho, k) = \mathcal{A}_m(\rho, k)$ for all nonfaulty processors p_m, the same is true for all nonfaulty processors. As a result, all nonfaulty processors must choose to perform a_i simultaneously at time ℓ. Third, if ρ satisfies $\mathit{pro}(a_i)$, then the run ζ of any protocol implementing $\mathcal{C}$ corresponding to ρ must satisfy $\mathit{pro}(a_i)$, and hence a_i must be performed in ζ. As we have already seen, an action must also be performed in ρ. Since $\mathit{pro}(a_i) \supset \neg \mathit{enabled}(a_j)$ for all $j \neq i$, the set $\mathcal{A}_j(\rho, k)$ of a nonfaulty processor p_j must contain no action other than a_i (if it contains any action at all). Thus, a_i must be the action performed in ρ. Finally, if ρ satisfies $\mathit{con}(a_i)$, then ρ does not satisfy $\mathit{enabled}(a_i)$, and no set $\mathcal{A}_j(\rho, \ell)$ for any nonfaulty processor p_j contains a_i. Thus, a_i is not performed in ρ. It follows that $\mathcal{F}_c$ implements $\mathcal{C}$. □

As a result of Theorem 7, we see that full-information protocols can be used as the basis of optimal protocols for simultaneous choice problems. Thus, we will restrict our attention to full-information protocols in the remainder of this paper: Unless otherwise specified, all protocols mentioned will be full-information protocols, and all runs will be runs of such protocols. More important, however, a consequence of Theorem 7 is that designing an optimal protocol for a simultaneous choice problem $\mathcal{C}$ essentially reduces to testing for common knowledge of certain facts: In order to design an optimal protocol for $\mathcal{C}$, it is enough to construct the tests for common knowledge of the facts $\mathit{enabled}(a_i)$. We note that the fundamental property of common knowledge underlying the existence of such tests is the fact that $C_{\mathcal{N}}\varphi \supset E_{\mathcal{N}}\varphi$ is valid; that is, when φ becomes common knowledge, the fact that φ is common knowledge will follow from the view of every nonfaulty processor. The problem of implementing such tests is the subject of the following section.

Before ending this section, however, we consider the size of messages required by a full-information protocol $\mathcal{F}$. Such a protocol requires processors to send their entire



a. $\mathcal{G}(\rho, 3)$ b. $\mathcal{G}_1(\rho, 3)$

Figure 1: Communication graphs.

view during every round. Since, strictly speaking, the size of a processor's view may be exponential in the number of rounds elapsed, this protocol seems to require processors to send messages of exponential length. We now show, however, that there is a simple, compact representation of a processor's view that may be sent instead. Consequently, it will be possible to implement all full-information protocols (and in particular the optimal protocol $\mathcal{F}_c$) in a communication-efficient way in all variants of the omissions model.

Given a run ρ of $\mathcal{F}$, the *communication graph* of ρ (see Figure 1) represents the messages delivered in ρ. It is a layered graph (with one layer corresponding to every natural number, representing time on the global clock) in which each processor is represented by one node in every layer. We denote the node representing processor p at time ℓ by $\langle p, \ell \rangle$. Edges connect nodes in adjacent layers, with an edge between $\langle p, k-1 \rangle$ and $\langle q, k \rangle$ iff a message from p is delivered to q during round k. The *labeled communication graph* is obtained by labeling the layer 0 nodes of the communication graph with processors' names and initial states, and by labeling the layer k nodes (for $k > 0$) with the requests the processors receive from external clients during round k. We note in passing that, since ρ is a run of the full-information protocol $\mathcal{F}$, its labeled communication graph uniquely determines its operating environment. For every point (ρ, ℓ), we denote by $\mathcal{G}(\rho, \ell)$ the first $\ell + 1$ layers of the labeled communication graph of ρ, representing the first ℓ rounds of ρ. For example, illustrated in Figure 1(a) is a graph $\mathcal{G}(\rho, 3)$ depicting the first 3 rounds of a run ρ. We say that $\mathcal{G}(\rho, \ell)$ has *length* ℓ.

Informally, at every point (ρ, ℓ) of a run of $\mathcal{F}$, a processor p_i's view corresponds to a certain subgraph $\mathcal{G}_i(\rho, \ell)$ of $\mathcal{G}(\rho, \ell)$. For example, the subgraph $\mathcal{G}_1(\rho, 3)$ of $\mathcal{G}(\rho, 3)$ is illustrated in Figure 1(b). We define the subgraph $\mathcal{G}_i(\rho, \ell)$ of $\mathcal{G}(\rho, \ell)$ inductively as follows. For $\ell = 0$ the subgraph $\mathcal{G}_i(\rho, 0)$ consists of the labeled node $\langle p_i, 0 \rangle$. For $\ell > 0$ the subgraph $\mathcal{G}_i(\rho, \ell)$ consists of the labeled node $\langle p_i, \ell \rangle$, the subgraph $\mathcal{G}_i(\rho, \ell - 1)$, the edges from layer $\ell - 1$ nodes to $\langle p_i, \ell \rangle$, and the subgraphs $\mathcal{G}_j(\rho, \ell - 1)$ for every layer $\ell - 1$

node $\langle p_j, \ell - 1\rangle$ adjacent to $\langle p_i, \ell\rangle$. Given a set S of processors, it is convenient to denote by $\mathcal{G}_S(\rho, \ell)$ the union of the graphs $\mathcal{G}_i(\rho, \ell)$ for every $p_i \in S$. We remark that $\mathcal{G}_S(\rho, \ell)$ uniquely determines $\mathcal{G}_i(\rho, \ell)$ for every $p_i \in S$. The next lemma states that a processor's view of the labeled communication graph uniquely determines its view of the run.

Lemma 8: Let ρ and ρ' be runs of a full-information protocol $\mathcal{F}$. For every processor p_i and time ℓ, $v(p_i, \rho, \ell) = v(p_i, \rho', \ell)$ iff $\mathcal{G}_i(\rho, \ell) = \mathcal{G}_i(\rho', \ell)$.

Proof: We proceed by induction on ℓ. The case of $\ell = 0$ is immediate. Suppose $\ell > 0$ and the inductive hypothesis holds for $\ell - 1$.

Suppose p_i has the same view at time ℓ in both ρ and ρ'. This implies, in particular, that p_i has the same view at time $\ell - 1$ in ρ and ρ', and from the inductive hypothesis it follows that $\mathcal{G}_i(\rho, \ell - 1) = \mathcal{G}_i(\rho', \ell - 1)$. In addition, this implies that p_i must receive the same input during round ℓ in ρ and ρ', and hence $\langle p_i, \ell\rangle$ is labeled with the same input in $\mathcal{G}_i(\rho, \ell)$ and $\mathcal{G}_i(\rho', \ell)$. If p_i does not receive a message from a processor p_j during round ℓ in ρ and ρ', then there is no edge from $\langle p_j, \ell - 1\rangle$ to $\langle p_i, \ell\rangle$ in either $\mathcal{G}_i(\rho, \ell)$ or $\mathcal{G}_i(\rho', \ell)$. If p_i *does* receive a message from a processor p_j during round ℓ in ρ and ρ', then it receives the same message in both runs, and p_j must have the same view at time $\ell - 1$ in both runs. Hence, there is an edge from $\langle p_j, \ell - 1\rangle$ to $\langle p_i, \ell\rangle$ in both $\mathcal{G}_i(\rho, \ell)$ and $\mathcal{G}_i(\rho', \ell)$, and by the inductive hypothesis we have that $\mathcal{G}_j(\rho, \ell - 1) = \mathcal{G}_j(\rho', \ell - 1)$. Thus, $\mathcal{G}_i(\rho, \ell) = \mathcal{G}_i(\rho', \ell)$.

Conversely, suppose $\mathcal{G}_i(\rho, \ell) = \mathcal{G}_i(\rho', \ell)$. It follows that $\mathcal{G}_i(\rho, \ell - 1) = \mathcal{G}_i(\rho', \ell - 1)$, and by the inductive hypothesis p_i has the same view at time $\ell - 1$ in ρ and ρ'. The node $\langle p_i, \ell\rangle$ must be labeled with the same input in $\mathcal{G}_i(\rho, \ell)$ and $\mathcal{G}_i(\rho', \ell)$, so p_i receives the same input during round ℓ in ρ and ρ'. The edges from layer $\ell - 1$ nodes to $\langle p_i, \ell\rangle$ are the same in $\mathcal{G}_i(\rho, \ell)$ and $\mathcal{G}_i(\rho', \ell)$, so p_i receives messages from the same processors during round ℓ in ρ and ρ'. Again, $\mathcal{G}_j(\rho, \ell - 1) = \mathcal{G}_j(\rho', \ell - 1)$ for every node $\langle p_j, \ell - 1\rangle$ adjacent to $\langle p_i, \ell\rangle$, and by the inductive hypothesis p_j has the same view at time $\ell - 1$ in ρ and ρ'. Since a protocol is a deterministic function of processor views, p_i receives the same messages during round ℓ in ρ and ρ'. It follows that p_i has the same view at time ℓ in both ρ and ρ'. □

Lemma 8 implies that a processor's view of the run and a processor's view of the corresponding labeled communication graph convey the same information: Given either the graph $\mathcal{G}_i(\rho, \ell)$ or the view $v(p_i, \rho, \ell)$, reconstructing the other is straightforward. Therefore, an equivalent implementation of a full-information protocol requires the processors to send the labeled communication graphs corresponding to their views instead of sending their complete views. From now on, we will use the term *full-information protocol* to refer to this equivalent form. It is easy to see that the size of $\mathcal{G}_i(\rho, \ell)$ is polynomial in the number of processors n, the global time ℓ, and the size of the requests received from external clients. It follows that messages required by a full-information protocol are of polynomial size.[5] Furthermore, given the labeled communication graphs corresponding to the views at time $\ell - 1$ of the processors that send messages to a given processor p_i during round ℓ, it is easy to construct the labeled communication graph corresponding to p_i's view at time ℓ. Thus, the use of such compact representations of a processor's view is computationally efficient as well as communication efficient. Finally, recall that we have formally defined a test for common knowledge to be a function of processor views (as well at n and t). In light of the preceding discussion, there is no loss of generality in assuming that such a test is a function of communication graphs corresponding to processor views. We now turn to the problem of implementing such tests.

6 Testing for Common Knowledge

The previous section established the claim that tests for common knowledge provide a very powerful programming technique: The design of optimal protocols for simultaneous choice problems reduces to implementing tests for common knowledge of certain facts. In this section we investigate the problem of implementing tests for common knowledge in the different variants of the omissions model. With such tests, we will be able to construct optimal protocols for simultaneous choice problems in these models. As we will see, properties of the different variants of the omissions model cause dramatic differences in the complexity of testing for common knowledge. In addition, the optimal protocols we construct will have interesting properties that vary according to the failure model.

Recall that a protocol is a function that, given the number of processors n, the bound t on the number of faulty processors, and a processor's view, yields a list of the actions the processor should perform, as well as the messages it should send in the next round. (Thus, the protocols we are interested in are uniform in n and t.) Since the protocols we will be concerned with are full-information protocols, processors' views will be efficiently representable by labeled communication graphs. We will soon restrict our attention to simultaneous choice problems in which the external requests are of constant size. This restriction implies that processors' views at time ℓ will be of size polynomial in n and ℓ. A protocol will therefore determine the messages and actions required at time ℓ based on input of size polynomial in n and ℓ. Consequently, we will measure the complexity of computations performed by protocols at time ℓ in systems of n processors as a function of n and ℓ: By polynomial time, polynomial space, etc., we will mean polynomial in n and ℓ.

The definition of simultaneous choice problems presented in Section 4 is very general. So general, in fact, that it is possible to define simultaneous choice problems with a variety of anomalous properties. For example, it is possible to define a simultaneous choice problem in which $pro(a)$ is the fact $\varphi =$ "the first round in which p receives an

[5] In the Byzantine failure models in which processors are allowed to lie (or maliciously deviate from the protocol), however, such compact representations are not guaranteed to exist; cf. [C].

external request is a round whose number is the index of a halting Turing machine" (in some a priori well-defined enumeration of Turing machines), and *con*(a) is $\neg\varphi$. Clearly, since it is undecidable whether φ holds even given the view of p after it receives its first request, it will also be undecidable which of $C_{\mathcal{N}}\varphi$ or $C_{\mathcal{N}}\neg\varphi$ holds when processor p's view becomes common knowledge. It follows that this simultaneous choice problem cannot be effectively implemented by a computable protocol. Similarly, one can construct simultaneous choice problems in which evaluation of the conditions is intractable, rather than undecidable as in the above example. It is also possible to introduce anomalies by defining the sets Γ_i of external inputs in strange ways. Since we are not interested in problems involving such inherent anomalies, we will avoid them by making restrictions on the relevant facts and the inputs arising in the simultaneous choice problems we will consider in the sequel.

We first define the class of *practical* facts, which will be used to restrict the conditions that specify a simultaneous choice problem. Roughly speaking, one essential property of a practical fact φ is that it is easy to determine from a processor's view whether a run satisfies φ. More formally, we denote by "$\mathcal{G}_S(\rho,\ell)$" the property of being a run ρ' such that $\mathcal{G}_S(\rho,\ell) = \mathcal{G}_S(\rho',\ell)$. Consequently, if $\mathcal{G}_S(\rho,\ell) \supset \varphi$ is valid in a system, then every run ρ' of the system satisfying $\mathcal{G}_S(\rho,\ell) = \mathcal{G}_S(\rho',\ell)$ must also satisfy φ. In this case, we say that $\mathcal{G}_S(\rho,\ell)$ *determines* φ. Notice, for example, that no finite labeled communication graph $\mathcal{G}_S(\rho,\ell)$ can determine that a run is failure-free (since the run is infinite, and a failure can always happen outside the finite scope depicted by the graph). With this notion in mind, a fact φ is said to be *practical* within a class of systems $\{\Sigma(n,t) : n \geq t+2\}$ if the following conditions hold: (i) φ is a fact about the input and the existence of failures, and (ii) there is a polynomial-time algorithm to determine, given n, t, and a graph $\mathcal{G}_S(\rho,\ell)$ of a point of $\Sigma(n,t)$, whether $\mathcal{G}_S(\rho,\ell) \supset \varphi$ is valid in $\Sigma(n,t)$. The first condition is justified by the fact that we will be testing for common knowledge of the conditions *enabled*(a_i) arising from natural simultaneous choice problems, and such conditions are typically conditions on the input and existence of failures. The second condition ensures that it is easy to test whether a labeled communication graph determines that the fact holds. (We make this restriction since it would clearly be unreasonable to expect the processors to be able to efficiently identify and act based on facts that are intractable to compute from the labeled communication graph.)

We now consider a natural restriction on the sets Γ_i of possible inputs. A class of systems is said to be *practical* if there are two fixed finite sets S and M of initial states and external requests, respectively, such that each Γ_i in all systems of the class is the set of complete input histories whose initial state is in S, and in which the input received in every round is a subset of M. This condition ensures that the input sets are of a simple form. In particular, it implies that all Γ_i's are identical, and that the input received by a processor during any given round is of constant size.

Having defined the notions of practical facts and practical classes of systems, we say that a simultaneous choice $\mathcal{C}$ is *practical* if (i) the class of systems determined by a full-information protocol and $\mathcal{C}$ is practical, and (ii) each condition *enabled*(a_i) is practical within this class of systems. Essentially all natural simultaneous choice problems are practical. In particular, all simultaneous choice problems appearing in the literature are practical. Our analysis will hence be restricted to testing for common knowledge of practical facts and to designing and implementing optimal protocols for practical simultaneous choice problems. We remark, however, that our analysis will apply to a more general class of simultaneous choice problems, whose precise characterization is somewhat complicated.

In Section 5 we programmed protocols for simultaneous choice problems in a high-level language in which processors' actions depend on explicit tests for common knowledge. Recall that *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) is a test nonfaulty processors can use to determine whether *enabled*(a_i) is common knowledge: Given the graph corresponding to the view of a nonfaulty processor at (ρ,ℓ) as input, *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) returns **true** iff $(\rho,\ell) \models C_{\mathcal{N}}\mathit{enabled}(a_i)$. Theorem 7 implies that given such a test for each condition *enabled*(a_i), the protocol $\mathcal{F}_c$ is an optimal protocol for $\mathcal{C}$. Until this point, however, we have sidestepped the issue of whether such tests actually exist. With the next lemma we see that, for practical simultaneous choice problems, such tests can be implemented in polynomial space.

Lemma 9: If $\mathcal{C}$ is a practical simultaneous choice problem, then for each action a_i the test *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) can be implemented in polynomial space.

Proof: We must prove the existence of an algorithm *test-for-*$C_{\mathcal{N}}$*enabled*(a_i) determining in polynomial space whether *enabled*(a_i) is common knowledge at (ρ,ℓ), given as input n, t, and the graph $\mathcal{G}_j(\rho,\ell)$ corresponding to the view of a nonfaulty processor p_j at (ρ,ℓ). We will actually exhibit a nondeterministic, polynomial-space algorithm A_i determining whether *enabled*(a_i) is *not* common knowledge at (ρ,ℓ). Since NPSPACE=PSPACE and PSPACE is closed under complementation (see [HU]), the existence of the algorithm A_i implies the existence of an algorithm *test-for-*$C_{\mathcal{N}}$*enabled*(a_i). Let $\{\Sigma(n,t) : n \geq t+2\}$ be a class of systems determined by a full-information protocol and the problem $\mathcal{C}$. We claim that such an algorithm A_i need only guess a point (η,ℓ), and show both that $(\rho,\ell) \sim (\eta,\ell)$ and that $\mathcal{G}(\eta,\ell) \supset \mathit{enabled}(a_i)$ is *not* valid in $\Sigma(n,t)$. To see this, notice that since $\mathcal{G}(\eta,\ell) \supset \mathit{enabled}(a_i)$ is *not* valid in the system, there must be a point (η',ℓ) such that $\mathcal{G}(\eta,\ell) = \mathcal{G}(\eta',\ell)$ and $(\eta',\ell) \not\models \mathit{enabled}(a_i)$. Construct the run with the input of η' in which processors fail precisely as they do in η for the first ℓ rounds, and in which no processor fails after time ℓ. Let ζ be a run obtained by adding to this run a single failure after time ℓ iff there is a failure in η'. Since ζ and η' must satisfy the same facts about the input and existence of failures, $(\eta',\ell) \not\models \mathit{enabled}(a_i)$ implies $(\zeta,\ell) \not\models \mathit{enabled}(a_i)$. Since at least one nonfaulty processor in η is nonfaulty in ζ, and also has the same view at time ℓ since $\mathcal{G}(\zeta,\ell) = \mathcal{G}(\eta,\ell)$, we have $(\eta,\ell) \sim (\zeta,\ell)$. Therefore, $(\rho,\ell) \sim (\zeta,\ell)$ and $(\zeta,\ell) \not\models \mathit{enabled}(a_i)$, and it follows that $(\rho,\ell) \not\models C_{\mathcal{N}}\mathit{enabled}(a_i)$.

We now describe the algorithm A_i in greater detail. Notice that since $\mathcal{C}$ is practical, the input received by a processor in every round of a run of $\Sigma(n,t)$ is of constant size, and hence it is possible to construct the labeled communication graph of any point of $\Sigma(n,t)$ in polynomial space. The algorithm A_i begins by constructing the graph $\mathcal{G}(\rho',\ell)$ of a run ρ' by adding to the graph $\mathcal{G}_j(\rho,\ell)$ received as input all edges not recorded as missing in $\mathcal{G}_j(\rho,\ell)$. Notice that since p_j is nonfaulty in ρ, it is nonfaulty in ρ' as well, and hence $(\rho,\ell) \sim (\rho',\ell)$. The algorithm A_i then shows that $(\rho',\ell) \sim (\eta,\ell)$ (and hence that $(\rho,\ell) \sim (\eta,\ell)$) in polynomial space by constructing one by one the graph $\mathcal{G}(\zeta_i,\ell)$ of each point (ζ_i,ℓ) in a path from (ρ',ℓ) to (η,ℓ) in the similarity graph. For each pair of points (ζ_{i-1},ℓ) and (ζ_i,ℓ), the algorithm shows that some nonfaulty processor p_k has the same view at both points by choosing p_k, exhibiting for each point an assignment of faulty processors (consistent with their respective graphs) in which p_k is nonfaulty, and showing that p_k has the same view at both points by verifying $\mathcal{G}_k(\zeta_{i-1},\ell) = \mathcal{G}_k(\zeta_i,\ell)$. Finally, since *enabled*(a_i) is a practical fact, A_i can show in polynomial time (and hence in polynomial space) that $\mathcal{G}(\eta,\ell) \supset$ *enabled*(a_i) is *not* valid in the system $\Sigma(n,t)$. □

It is important to realize that Lemma 9 holds in all variants of the omissions model: The failure model is a parameter of a simultaneous choice problem, and we have made no assumptions restricting the failure model in this result. We note that the proof of Lemma 9 actually shows that testing for common knowledge of any practical fact can be done in polynomial space. In fact, the proof shows that such tests have effective implementations even when the algorithm determining whether $\mathcal{G}(\rho,\ell) \supset$ *enabled*(a_i) is valid does not run in polynomial time (although the problem must still be decidable). In this case, however, the test is guaranteed to run in polynomial space only if this computation can be performed using polynomial space. The most important consequence of Lemma 9, however, is that practical simultaneous choice problems have polynomial-space optimal protocols.

Theorem 10: If $\mathcal{C}$ is an implementable practical simultaneous choice problem, then there is a polynomial-space, optimal protocol for $\mathcal{C}$.

With Theorem 10 we see that practical simultaneous choice problems do have effective optimal protocols. In general, however, connected components in the similarity graph may be of exponential size, and paths in such components may be of exponential length. It therefore follows that the polynomial-space protocol given by Theorem 10 requires the processors to perform exponential-time computations between consecutive rounds of communication. The resulting protocol is therefore clearly not a reasonable protocol to use in practice. A crucial question at this point is whether there are *efficient* optimal protocols for simultaneous choice problems. Recall that we have already seen that optimal protocols can be implemented in a way that makes efficient use of communication. The rest of the paper is devoted to investigating ways of implementing tests for common knowledge in variants of the omissions model in a computationally-efficient manner, and therefore of implementing efficient, optimal protocols for simultaneous choice problems in these models.

6.1 The Omissions Model

In this section we consider the problem of efficiently implementing tests for common knowledge in the omissions failure model. In particular, we develop a construction that crisply characterizes the connected component of a point in the similarity graph. This construction determines a subgraph of the labeled communication graph with the property that two points are similar iff their respective subgraphs are identical. As stated in Theorem 2, the connected component of a point in the similarity graph completely determines what facts are common knowledge at that point. As a result, this construction enables us to devise efficient tests for common knowledge, and hence efficient protocols for simultaneous choice problems that are optimal in all runs.

Dwork and Moses address in [DM] the problem of implementing tests for common knowledge in the *crash failure model.* In the crash failure model, processors fail by crashing; that is, faulty processors may successfully send messages to some processors during their failing round, but will not successfully send any messages in any later round. As a result, a faulty processor is "out of the game" after its failing round, and no longer contributes to the knowledge of the remaining processors. The analysis performed by Dwork and Moses focuses on the notion of a *clean round*, a round in which no processor crashes. In runs of a full-information protocol, a clean round ensures that all nonfaulty processors receive the same set of messages. After such a round, all nonfaulty processors have an identical view of the part of the run that precedes the clean round. Dwork and Moses show that facts about the initial configuration become common knowledge exactly when it becomes common knowledge a clean round has occurred. Dwork and Moses complete their analysis by characterizing when this happens. In the omissions model, however, a faulty processor may continue to contribute to the knowledge of the nonfaulty processors, even after its first failing round, since it may fail intermittently in later rounds as well. The situation is therefore more complicated, and clean rounds no longer play the same role here as they do in the crash failure model. Furthermore, to the best of our understanding, there is no direct analog to the notion of a clean round in the omissions model. The approach used by Dwork and Moses in the crash failure model, therefore, does not seem to extend to this model. As a result, we are forced to take a different approach.

Our approach to the problem of testing for common knowledge during runs of a full-information protocol is motivated by a careful analysis of what facts do *not* become common knowledge.[6] (Unless otherwise mentioned, all protocols referred to in this sec-

[6] As mentioned in the introduction, since the technical details of the proofs in this section may make it difficult to obtain a high-level understanding of our approach, we encourage the reader to skip the proofs on the first reading.

tion will be full-information protocols.) We begin with a technical result, similar to Lemma 15 of [DM], saying that simple modifications of a processor's faulty behavior preserve similarity, and hence what facts are common knowledge. Throughout the remainder of this paper it will be convenient to refer to runs differing only in some aspect of their operating environments. Given two runs ρ and ρ' of a protocol $\mathcal{F}$, we will say that ρ differs from ρ' *only in* a certain aspect of the operating environment if ρ is the result of executing $\mathcal{F}$ in an operating environment that differs from that of ρ' only in the said aspect. Notice that while their operating environments may be similar, the messages sent in the two runs may actually be quite different. We say that a processor is *silent* from time k if it fails to send every message in every round following time k.

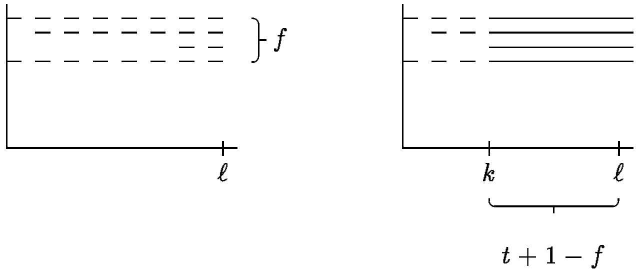
Lemma 11: Let ρ and ρ' be runs differing only in the (faulty) behavior displayed by processor p after time k, and suppose no more than f processors fail in either ρ or ρ'. If $\ell - k \leq t + 1 - f$, then $(\rho, \ell) \sim (\rho', \ell)$.

Proof: If $k \geq \ell$ then $\mathcal{G}(\rho, \ell) = \mathcal{G}(\rho', \ell)$, and Lemma 8 implies that $(\rho, \ell) \sim (\rho', \ell)$. Therefore, assume $k < \ell$. We proceed by induction on $j = \ell - k$. Without loss of generality, we may assume that ρ and ρ' actually differ in the faulty behavior of p, and hence that p fails in one of these runs. Notice that since p already fails in one of these runs and yet no more than f processors fail in either run, it is clear that at most $f \leq t$ processors fail in any run differing from either run only in the faulty behavior of p.

Suppose $j = 1$ (that is, $k = \ell - 1$). Since $t \leq n - 2$ and since ρ and ρ' differ only in the behavior of p, there are two processors q and r (other than p) that do not fail in either run. Let ρ_r be the run differing from ρ only in that p sends to r during round ℓ of ρ_r iff it does so in ρ' (and notice that ρ_r may actually be equal to ρ). Since q's view at time ℓ is independent of whether p sends to r during round ℓ, we have $(\rho, \ell) \sim (\rho_r, \ell)$. Since $\mathcal{G}(\rho_r, \ell)$ and $\mathcal{G}(\rho', \ell)$ differ only in the messages that p sends to processors other than r in round ℓ, and r's view at (ρ_r, ℓ) is independent of whether p sends to the remaining processors during round ℓ, we have $(\rho_r, \ell) \sim (\rho', \ell)$. Thus, by the transitivity of "$\sim$," we have $(\rho, \ell) \sim (\rho', \ell)$.

Suppose $j > 1$ (that is, $k < \ell - 1$) and the inductive hypothesis holds for $j - 1$. Let ρ_i be the run differing from ρ only in that for each processor q in $\{p_1, \ldots, p_i\}$ processor p sends to q during round $k + 1$ in ρ_i iff it does so in ρ'. Notice that $\rho = \rho_0$ and $\rho' = \rho_n$. We will show that $(\rho, \ell) \sim (\rho_i, \ell)$ for all $i \geq 0$. Since ρ_n differs from ρ' only in the faulty behavior of p after time $k + 1$, and since $\ell - (k + 1) = j - 1$, it will follow by the inductive hypothesis for $j - 1$ that $(\rho_n, \ell) \sim (\rho', \ell)$. Finally, by the transitivity of "$\sim$," we will have $(\rho, \ell) \sim (\rho', \ell)$ as desired.

We now proceed by induction on i to show that $(\rho, \ell) \sim (\rho_i, \ell)$ for all $i \geq 0$. The case of $i = 0$ is trivial. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $(\rho, \ell) \sim (\rho_{i-1}, \ell)$. Notice ρ_{i-1} and ρ_i differ at most in whether p sends a message to p_i during round $k+1$. Let η be the run differing from ρ_{i-1} in that p_i is silent from time $k+1$



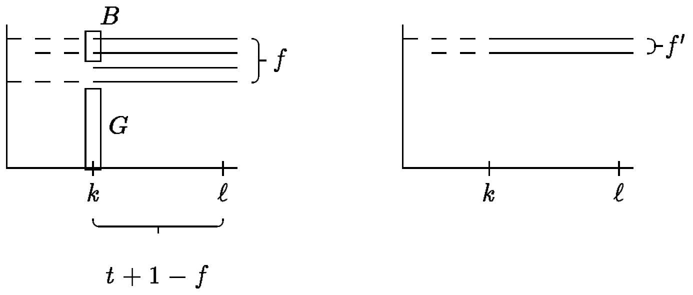
The run ρ_1. The run ρ_2.

Figure 2: Runs illustrating Lemma 12.

in η. Suppose no more than g processors fail in either ρ_{i-1} or η. Notice that $g \le f+1$. Therefore, since $1 < \ell - k \le t+1-f$ we have $f < t$ and $g \le t$, so at most t processors fail in η. Furthermore, $\ell - (k+1) \le t+1-(f+1) \le t+1-g$. Since, in addition, ρ_{i-1} and η differ only in the faulty behavior of p_i after time $k+1$, the inductive hypothesis for $j-1$ implies $(\rho_{i-1}, \ell) \sim (\eta, \ell)$. Now, since p_i is silent from time $k+1$ in η, the view of a nonfaulty processor at (η, ℓ) is independent of whether p sends to p_i during round $k+1$, so $(\eta, \ell) \sim (\eta', \ell)$ where η' differs from η in that p sends to p_i during round $k+1$ in η' iff it does so in ρ_i. Again, the inductive hypothesis for $j-1$ implies that $(\eta', \ell) \sim (\rho_i, \ell)$. By the transitivity of "$\sim$," it follows that $(\rho, \ell) \sim (\rho_i, \ell)$. □

While Lemma 11 is a technical lemma in the context of this work, it has a number of interesting consequences in its own right. In particular, the $(t+1)$-round lower bound on the number of rounds required for simultaneous Byzantine agreement is an immediate corollary of this lemma. The resulting proof of this lower bound is perhaps the simplest to appear in the literature (see [DM] for details). More important, however, is the fact that two corollaries of Lemma 11 enable us to characterize the connected components of the similarity graph. Consider the runs ρ_1 and ρ_2 of Figure 2, where we indicate only faulty behavior: solid lines indicate silence, and dashed lines indicate sporadic faulty behavior. Notice that f processors fail in ρ_1. In the following lemma we show that $(\rho_1, \ell) \sim (\rho_2, \ell)$ where ρ_2 differs from ρ_1 only in that processors failing in ρ_1 are silent in ρ_2 from time k, where $k = \ell - (t+1-f)$. This implies, for instance, that the views at time k of processors failing in ρ_1 are not common knowledge at time ℓ since these processors are silent from time k in ρ_2.

Lemma 12: Let ρ_1 be a run in which f processors fail. Let ρ_2 be the run differing from ρ_1 only in that processors failing in ρ_1 are silent from time k in ρ_2, where $k = \ell - (t+1-f)$. Then $(\rho_1, \ell) \sim (\rho_2, \ell)$.



The run ρ_2.

The run ρ_3.

Figure 3: Runs illustrating Lemma 13.

Proof: Let $q_1, \ldots, q_f$ be the faulty processors in ρ_1. Let η_i be the run differing from ρ_1 in that processors $q_1, \ldots, q_i$ are silent from time k in η_i. Notice that $\rho_1 = \eta_0$ and $\rho_2 = \eta_f$. We proceed by induction on i to show that $(\rho_1, \ell) \sim (\eta_i, \ell)$ for all i. The case of $i = 0$ is trivial. Suppose $i \geq 0$ and the inductive hypothesis holds for $i - 1$; that is, $(\rho_1, \ell) \sim (\eta_{i-1}, \ell)$. Since η_{i-1} and η_i differ at most in the faulty behavior of q_i after time k, it follows by Lemma 11 that $(\eta_{i-1}, \ell) \sim (\eta_i, \ell)$. By the transitivity of "$\sim$," we have $(\rho_1, \ell) \sim (\eta_i, \ell)$. □

Before discussing the second lemma, we make an important definition. Given a point (ρ, k) and a set of processors G, let

$$B(G, \rho, k) \overset{\text{def}}{=} \{p : (\rho, k) \models I_G(\text{“}p \textit{ is faulty”})\}\,.$$

By this definition, $B(G, \rho, k)$ is the set of processors implicitly known by G at (ρ, k) to be faulty. An important property of the omissions failure model is that processors fail *only* by failing to send messages. It follows that G implicitly knows at (ρ, k) that a processor p is faulty iff G implicitly knows at (ρ, k) of some processor q not receiving a message from p before time k; that is, $\mathcal{G}_G(\rho, k)$ contains no edge from $\langle p, \ell - 1\rangle$ to $\langle q, \ell\rangle$ for some node $\langle q, \ell\rangle$ of $\mathcal{G}_G(\rho, k)$. It is therefore simple and straightforward to compute $B(G, \rho, k)$ given $\mathcal{G}_G(\rho, k)$.

The essence of the second lemma is captured by the runs ρ_2 and ρ_3 of Figure 3. In the run ρ_2, the f faulty processors are silent from time $k = \ell - (t + 1 - f)$. The set G is the set of nonfaulty processors and $B = B(G, \rho_2, k)$. The run ρ_3 differs from ρ_2 only in that processors in $P - B$ do not fail in ρ_3. The following lemma states that $(\rho_2, \ell) \sim (\rho_3, \ell)$. This implies, for instance, that the failure of processors in $P - B$ cannot be common knowledge at (ρ_2, ℓ) since they do not fail in ρ_3. Formally, we have (see Figure 3):

Lemma 13: Let ρ_2 be a run in which the f faulty processors are silent from time $k = \ell - (t + 1 - f)$. Let G be the set of nonfaulty processors in ρ_2, and let $B = B(G, \rho_2, k)$.

Let ρ_3 be the run differing from ρ_2 only in that processors in $P - B$ do not fail. Then $(\rho_2, \ell) \sim (\rho_3, \ell)$.

Proof: If a processor p in $P - B$ fails to a processor q during some round $j \leq k$ of ρ_2 (in which case it must be that $p \in P - B - G$), then the node $\langle q, j \rangle$ must not be a node of $\mathcal{G}_G(\rho_2, k)$ or the failure of p would be implicitly known by G at time k and p would be in B, a contradiction. Thus, $\mathcal{G}_G(\rho_2, k)$ is independent of whether $\mathcal{G}(\rho_2, k)$ contains an edge from p to q during round j. Let ρ_2' be a run differing from ρ_2 only in that no processor in $P - B$ fails *before* time k in ρ_2'. By the previous discussion, $\mathcal{G}_G(\rho_2, k) = \mathcal{G}_G(\rho_2', k)$. In both ρ_2 and ρ_2' every processor in G successfully sends every message after time k and every processor in $P - G$ is silent from time k. Since, in addition, every processor in G receives the same input after time k in ρ_2 and ρ_2', we have $\mathcal{G}_G(\rho_2, \ell) = \mathcal{G}_G(\rho_2', \ell)$. Given that G is the set of nonfaulty processors in ρ_2, each of which is also nonfaulty in ρ_2', it follows by Lemma 8 that $(\rho_2, \ell) \sim (\rho_2', \ell)$. Since the runs ρ_2' and ρ_3 differ only in the faulty behavior of processors in $P - B$ after time k, by repeated application of Lemma 11 it follows that $(\rho_2', \ell) \sim (\rho_3, \ell)$. Hence, $(\rho_2, \ell) \sim (\rho_3, \ell)$. □

Having seen Lemmas 12 and 13, let us consider how these results suggest a characterization of the similarity graph, and hence of what facts are common knowledge at a given point. Going back to Figures 2 and 3, notice that if $f' < f$ (which implies, referring to Figure 3, that not all f processors failing in ρ_1 are implicitly known at time $k = \ell - (t + 1 - f)$ to be faulty), then by setting $\rho_1' = \rho_3$ we can apply Lemmas 12 and 13 again (this time starting from ρ_1' instead of ρ_1). Iterating this process, we reach a run $\hat{\rho}$ satisfying $(\rho_1, \ell) \sim (\hat{\rho}, \ell)$, where the $\hat{f}$ processors failing in $\hat{\rho}$ are silent from time $\hat{k} = \ell - (t + 1 - \hat{f})$, and where all faulty processors are implicitly known to be faulty by the nonfaulty processors at $(\hat{\rho}, \hat{k})$. This run $\hat{\rho}$ is a fixpoint of this iterative process; setting $\hat{\rho}_1 = \hat{\rho}$, the runs $\hat{\rho}_2$ and $\hat{\rho}_3$ constructed in Lemmas 12 and 13 are identical to $\hat{\rho}$. It is the joint view of the nonfaulty processors at $(\hat{\rho}, \hat{k})$, we claim, that characterizes the connected component of (ρ_1, ℓ) in the similarity graph, and hence what facts are common knowledge at (ρ_1, ℓ). In order to make this claim precise, we now define a local version of this iterative process, illustrated in Figure 4, that processors can use to construct this joint view locally.

Let p be an arbitrary processor. We define $G_0 = \{p\}$ and $k_0 = \ell$, and we define G_{i+1} and k_{i+1} inductively as follows. Denoting $B(G_i, \rho, k_i)$ by B_i, let

$$G_{i+1} = P - B_i$$

$$k_{i+1} = \ell - (t + 1 - |B_i|).$$

Notice that when $k_{i+1} < 0$, the view at time k_{i+1} of every processor in G_{i+1} is the distinguished *empty* view, and hence B_{i+1} must be empty. As a consequence, for all $j > i + 1$, we have that $G_j = P$, $k_j = \ell - (t + 1)$, and B_j is empty. This construction determines three (infinite) sequences $\{G_i\}$, $\{k_i\}$, and $\{B_i\}$. In the next few pages we will

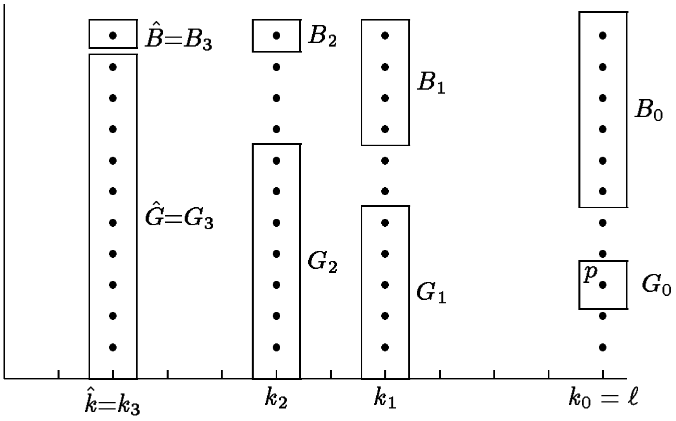


Figure 4: An example of the construction when $t = 9$.

see that these sequences have limits $\hat{G}$, $\hat{k}$, and $\hat{B}$, and that these limits are independent of the processor with which the construction is begun. As a result, individual processors will be able to construct these values based solely on their local view. We will see that the joint view of $\hat{G}$ at time $\hat{k}$ completely characterizes the connected component of (ρ, ℓ) in the similarity graph, and hence what facts are common knowledge at (ρ, ℓ). This construction will therefore provide an efficient way of determining what facts are common knowledge at a given point.

Among other things, this construction captures a number of essential aspects of the information flow during the run up to time ℓ. In particular, one important property of this construction is the following:

Lemma 14: Every processor in G_{i+1} successfully sends to every processor in G_i in every round before time k_i.

Proof: Suppose some processor q of G_{i+1} fails to send to a processor q' of G_i during a round before time k_i. Then q's failure to q' is implicitly known by G_i at time k_i, so $q \in B_i$ and $q \notin G_{i+1}$, a contradiction. □

One consequence of Lemma 14 is that the view of the processor p at time ℓ must contain the view of every processor in G_i at time k_i for every $i \geq 0$. Thus, one essential property of the construction is that it depends *only* on the view of processor p at (ρ, ℓ), and hence that p is able to perform the construction locally. A second essential property of the construction is that it converges within $t+1$ iterations, as we see with the following result.

Lemma 15: $\lim_{i\to\infty} G_i = G_{t+1}$ and $\lim_{i\to\infty} k_i = k_{t+1}$.

Proof: We will show that $B_{i+1} \subseteq B_i$ for all $i \geq 0$. Since B_0 contains at most t processors, it will then follow that there must be an $i \leq t$ for which $B_i = B_{i+1}$. From the definition of the construction, it is easy to see that we will have $B_i = B_{i+j}$ for all $j \geq 0$. In addition, we will have $G_{i+1} = G_{i+1+j}$ and $k_{i+1} = k_{i+1+j}$ for all $j \geq 0$, and we will be done. We proceed by induction on i. If $k_{i+1} < 0$, then B_{i+1} is empty and $B_{i+1} \subseteq B_i$, so let us assume $k_{i+1} \geq 0$. Suppose $i = 0$. By Lemma 14, every processor in G_1 must send to every processor in G_0 during round $k_1 + 1$. It follows that any failure implicitly known by G_1 at time k_1 must be implicitly known by G_0 at time k_0. Thus, $B_1 \subseteq B_0$. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $B_i \subseteq B_{i-1}$. If $B_i = B_{i-1}$, then $B_{i+1} = B_i$. If $B_i \subset B_{i-1}$, then $k_{i+1} < k_i$. By Lemma 14, G_{i+1} sends to G_i during round $k_{i+1} + 1$, so $B_{i+1} \subseteq B_i$. □

We denote the results of the construction (the limits of the sequences $\{G_i\}$, $\{k_i\}$, and $\{B_i\}$) by $\hat{G}$, $\hat{k}$, and $\hat{B}$. We denote these values by $\hat{G}(p, \rho, \ell)$, $\hat{k}(p, \rho, \ell)$, and $\hat{B}(p, \rho, \ell)$ when the processor p and the point (ρ, ℓ) are not clear from context. We now show, however, that these values are independent of processor p.

Lemma 16: For all processors p and q, $\hat{G}(p, \rho, \ell) = \hat{G}(q, \rho, \ell)$ and $\hat{k}(p, \rho, \ell) = \hat{k}(q, \rho, \ell)$.

Proof: We prove the claim by showing that $\hat{B}(p, \rho, \ell) = \hat{B}(q, \rho, \ell)$. Given that B_i uniquely determines G_{i+1} and k_{i+1}, this will imply the desired result. It suffices to show that $\hat{B}(p, \rho, \ell) \subseteq \hat{B}(q, \rho, \ell)$, since the other direction will follow by symmetry. Denote the intermediate results of the construction from the point (ρ, ℓ) starting with the processor p by G_i, k_i, and B_i, and the final results by $\hat{G}$, $\hat{k}$, and $\hat{B}$. Similarly, denote the intermediate results of the construction starting with q by G'_i, k'_i, and B'_i, and the final results by $\hat{G}'$, $\hat{k}'$, and $\hat{B}'$. We now show that $\hat{B} \subseteq \hat{B}'$. If $\hat{k} < 0$, then $\hat{B}$ is empty and $\hat{B} \subseteq \hat{B}'$, so assume $\hat{k} \geq 0$. We consider two cases. First, suppose $\hat{k} = \ell - 1$. In this case, $\hat{B}$ must contain t faulty processors since $\hat{k} = \ell - (t + 1 - |\hat{B}|)$. It follows that every processor in $\hat{G}$ must be nonfaulty in ρ and hence must send to G'_0 during round $\hat{k} + 1$, so $\hat{B} \subseteq B'_0$. Since, in addition, $|B'_0| \leq t$ and $|\hat{B}| = t$, we have $\hat{B} = B'_0$. It follows from the construction that $\hat{B} = B'_i$ for every $i \geq 0$, and hence that $\hat{B} = \hat{B}'$. Now, suppose $\hat{k} < \ell - 1$. Let r be an (arbitrary) nonfaulty processor in ρ. We claim that every processor g in $\hat{G}$ must send its view to r during round $\hat{k} + 1$. Suppose some processor g in $\hat{G}$ does not. Let j be the least integer such that $\hat{G} = G_j$. If $j = 1$, then r must send to G_0 during round $\hat{k} + 2$. If $j > 1$, then r must actually be a member of G_{j-1} since G_{j-1} must contain all of the nonfaulty processors. In either case, the failure of g to r during round $\hat{k} + 1$ must be implicitly known by G_{j-1} at time k_{j-1}, so $g \in B_{j-1}$. Since $\hat{G} = G_j = P - B_{j-1}$, we have $g \notin \hat{G}$, a contradiction. Thus, every processor in $\hat{G}$ must send to r during round $\hat{k} + 1$. We now proceed by induction on i to show that $\hat{B} \subseteq B'_i$ for all $i \geq 0$. Suppose $i = 0$. Every processor in $\hat{G}$ must send to the nonfaulty processor r during round $\hat{k} + 1$, and r must send to G'_0 during round $\hat{k} + 2$, so $\hat{B} \subseteq B'_0$. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $\hat{B} \subseteq B'_{i-1}$. If $\hat{B} = B'_{i-1}$, then $\hat{B} = B'_i$. If $\hat{B} \subset B'_{i-1}$, then $\hat{k} < k'_i$

since $\hat{k} = \ell - (t+1-|\hat{B}|)$ and $k'_i = \ell - (t+1-|B'_{i-1}|)$. Every processor in $\hat{G}$ must send to the nonfaulty processor r during round $\hat{k}+1$, and r must be contained in G'_i, so $\hat{B} \subseteq B'_i$. It follows that $\hat{B} \subseteq B'_i$ for all $i \geq 0$, and hence $\hat{B} \subseteq \hat{B}'$. □

As a result of Lemma 16, we see that $\hat{G}$, $\hat{k}$, and $\hat{B}$ depend only on the point (ρ, ℓ), and not the processor with which the construction begins. Thus, a third essential property of this construction is that *every* processor (and not just the nonfaulty processors) is able to compute locally the values of $\hat{G}$, $\hat{k}$, and $\hat{B}$. We will denote these values by $\hat{G}(\rho, \ell)$, $\hat{k}(\rho, \ell)$, and $\hat{B}(\rho, \ell)$ when (ρ, ℓ) is not clear from context. From the definition of the construction it is clear that the driving force behind the construction is the identity of the sets B_i. Notice that these sets are uniquely determined by the failure pattern, and do not depend on the run's input. Taking into account the input of a run, we are now in a position to show how the construction characterizes the connected components in the similarity graph. Denoting $\hat{G}(\rho, \ell)$ by $\hat{G}$ and $\hat{k}(\rho, \ell)$ by $\hat{k}$, we define

$$\hat{V}(\rho, \ell) \stackrel{\text{def}}{=} v(\hat{G}, \rho, \hat{k}).$$

This definition says that $\hat{V}(\rho, \ell)$ is the joint view of the processors in $\hat{G}(\rho, \ell)$ at time $\hat{k}(\rho, \ell)$. Our next lemma implies that $\hat{V}$ is the same at similar points, which implies that the joint view $\hat{V}(\rho, \ell)$ is common knowledge at (ρ, ℓ).

Lemma 17: If $(\rho, \ell) \sim (\rho', \ell)$ then $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$.

Proof: We proceed by induction on the distance d between the points (ρ, ℓ) and (ρ', ℓ). The case of $d = 0$ is trivial. Suppose that $d > 0$ and the inductive hypothesis holds for $d-1$. Since the distance between (ρ', ℓ) and (ρ, ℓ) is d, there must be a point (η, ℓ) whose distance from (ρ, ℓ) is $d-1$, and whose distance from (ρ', ℓ) is 1. The inductive hypothesis implies that $\hat{V}(\rho, \ell) = \hat{V}(\eta, \ell)$, and we must have $v(p, \eta, \ell) = v(p, \rho', \ell)$ for some processor p. As a consequence of Lemmas 14 and 16, the values of $\hat{V}(\eta, \ell)$ and $\hat{V}(\rho', \ell)$ depend only on the view of p at (η, ℓ) and (ρ', ℓ), respectively. Since p has the same view at (η, ℓ) and at (ρ', ℓ), we have $\hat{V}(\eta, \ell) = \hat{V}(\rho', \ell)$. Since $\hat{V}(\rho, \ell) = \hat{V}(\eta, \ell)$, it follows that $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$. □

One consequence of Lemma 17, together with Lemma 8 and the definition of $\hat{V}$ above, is that if $(\rho, \ell) \sim (\rho', \ell)$, then $\mathcal{G}_{\hat{G}}(\rho, \hat{k}) = \mathcal{G}_{\hat{G}}(\rho', \hat{k})$. We will find this a useful fact when proving the converse of Lemma 17; that is, that all points with the same $\hat{V}$ are similar, and hence that $\hat{V}$ completely characterizes the connected components of the similarity graph. Before we do so, however, we formalize the reasoning by which Lemmas 12 and 13 motivated consideration of the construction in the first place.

Lemma 18: Let ρ be a run, and let $\hat{G}$, $\hat{k}$, and $\hat{B}$ be the results of the construction from (ρ, ℓ). Let ρ' be the run differing from ρ only in that processors in $\hat{G}$ do not fail in ρ' and processors in $\hat{B}$ are silent from time $\hat{k}$ in ρ'. Then $(\rho, \ell) \sim (\rho', \ell)$.

Proof: Let G_i, k_i, and B_i be the intermediate results of the construction from (ρ, ℓ) starting with the nonfaulty processor p_j. For $i \geq 0$, define ρ_i to be the run differing from the run ρ only in that processors in B_i are silent from time k_i in ρ_i and the remaining processors do not fail in ρ_i. Notice that $\rho' = \rho_i$ for sufficiently large i. We proceed by induction on i to show that $(\rho, \ell) \sim (\rho_i, \ell)$ for all $i \geq 0$. Suppose $i = 0$. Since the subgraph $\mathcal{G}_j(\rho, \ell)$ must be independent of whether the graph $\mathcal{G}(\rho, \ell)$ is missing an edge from a processor in $P - B_0$ to a processor other than p_j, we have $\mathcal{G}_j(\rho, \ell) = \mathcal{G}_j(\rho_0, k_0)$. Since processor p_j is nonfaulty, it follows that $(\rho, \ell) \sim (\rho_0, \ell)$. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $(\rho, \ell) \sim (\rho_{i-1}, \ell)$. Lemma 12 implies $(\rho_{i-1}, \ell) \sim (\rho'_{i-1}, \ell)$ where ρ'_{i-1} differs from ρ_{i-1} in that processors in B_{i-1} (the processors failing in ρ_{i-1}) are silent from time k_i in ρ'_{i-1}. Lemma 13 implies $(\rho'_{i-1}, \ell) \sim (\rho_i, \ell)$. Thus, $(\rho, \ell) \sim (\rho_i, \ell)$. □

Finally, we have the following:

Lemma 19: If $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$ then $(\rho, \ell) \sim (\rho', \ell)$.

Proof: The fact $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$ implies $\hat{G}(\rho, \ell) = \hat{G}(\rho', \ell)$, $\hat{k}(\rho, \ell) = \hat{k}(\rho', \ell)$, and $\hat{B}(\rho, \ell) = \hat{B}(\rho', \ell)$. We therefore denote these values by $\hat{G}$, $\hat{k}$, and $\hat{B}$. Let ζ be a run that differs from ρ in that processors in $\hat{G}$ do not fail in ζ, and processors in $\hat{B}$ are silent from time $\hat{k}$ in ζ. Let ζ' be an analogous run with respect to ρ'. Lemma 18 implies that $(\rho, \ell) \sim (\zeta, \ell)$ and $(\rho', \ell) \sim (\zeta', \ell)$. In order to show that $(\rho, \ell) \sim (\rho', \ell)$, it is enough to show that $(\zeta, \ell) \sim (\zeta', \ell)$. Suppose $\hat{G} = \{q_1, \ldots, q_s\}$, and let ζ_i be the run differing from ζ in that $q_1, \ldots, q_i$ receive the same input after time $\hat{k}$ in ζ_i as they do in ζ'. We proceed by induction on i to show that $(\zeta, \ell) \sim (\zeta_i, \ell)$ for all $i \geq 0$. Since $\zeta = \zeta_0$, the case of $i = 0$ is trivial. Suppose $i > 0$ and the inductive hypothesis holds for $i - 1$; that is, $(\zeta, \ell) \sim (\zeta_{i-1}, \ell)$. Let η_{i-1} and η_i be runs differing from ζ_{i-1} and ζ_i, respectively, only in that q_i is silent from time $\hat{k}$ in η_{i-1} and η_i. Lemma 11 implies $(\zeta_{i-1}, \ell) \sim (\eta_{i-1}, \ell)$ and $(\zeta_i, \ell) \sim (\eta_i, \ell)$. In addition, since η_{i-1} and η_i differ only in the input received by q_i after time $\hat{k}$, and since q_i is silent from time $\hat{k}$ in both runs, we have $(\eta_{i-1}, \ell) \sim (\eta_i, \ell)$. Thus, $(\zeta, \ell) \sim (\zeta_i, \ell)$ for all $i \geq 0$. In particular, $(\zeta, \ell) \sim (\zeta_s, \ell)$. In order to complete the proof, it now suffices to show that $(\zeta_s, \ell) \sim (\zeta', \ell)$. Since $\mathcal{G}_{\hat{G}}(\rho, \hat{k}) = \mathcal{G}_{\hat{G}}(\rho', \hat{k})$, $(\rho, \ell) \sim (\zeta, \ell)$, and $(\rho', \ell) \sim (\zeta', \ell)$, Lemma 17 implies that $\mathcal{G}_{\hat{G}}(\zeta, \hat{k}) = \mathcal{G}_{\hat{G}}(\zeta', \hat{k})$. Notice that $\mathcal{G}_{\hat{G}}(\zeta_s, \hat{k}) = \mathcal{G}_{\hat{G}}(\zeta, \hat{k}) = \mathcal{G}_{\hat{G}}(\zeta', \hat{k})$. Notice, in addition, that processors in $\hat{G}$ do not fail in either ζ_s or ζ', and that the remaining processors (in $\hat{B}$) are silent from time $\hat{k}$ in both runs. Finally, notice that processors in $\hat{G}$ receive the same input after time $\hat{k}$ in both runs. It follows that $\mathcal{G}_{\hat{G}}(\zeta_s, \ell) = \mathcal{G}_{\hat{G}}(\zeta', \ell)$, and hence that $(\zeta_s, \ell) \sim (\zeta', \ell)$. Thus, $(\zeta, \ell) \sim (\zeta', \ell)$, as desired. □

Combining Lemmas 17 and 19 we see that $(\rho, \ell) \sim (\rho', \ell)$ iff $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$. We therefore have:

Theorem 20: $(\rho, \ell) \models C_{\mathcal{N}}\varphi$ iff $(\rho', \ell) \models \varphi$ for all ρ' satisfying $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$.

It follows that $\hat{V}(\rho,\ell)$ in a precise sense summarizes and uniquely determines the set of facts that are common knowledge at any given point (ρ,ℓ). The identity of $\hat{V}$ can be thought of as being composed of two components: The identity of $\hat{G}$ and $\hat{k}$, and the information about the input that is contained in the joint view $\hat{V}$. The definition of the construction implies that the identities of $\hat{G}$ and $\hat{k}$ depend only on the failure pattern, and hence carry only information about the failure pattern. The fact that $\hat{V}$ becomes common knowledge implies that certain information about the failure pattern must become common knowledge. It is difficult, however, to characterize the facts about the failure pattern that follow from the identity of $\hat{G}$ and $\hat{k}$. On the other hand, information about the input that follows from the views in $\hat{V}$ *does* characterize in a crisp way what facts about the input are common knowledge. Furthermore, it is easy to deduce from $\hat{V}$ whether the existence of a failure is common knowledge. As the following corollary will show, Theorem 20 implies that facts about the input and existence of failures that are common knowledge at the point (ρ,ℓ) must follow directly from the set $\hat{V}(\rho,\ell)$. We now make this statement precise. A run ρ, a set of processors G, and a time k determine a joint view $V = v(G,\rho,k)$. We denote by "V" the property of being a run in which the processors in G have the joint view V at time k (notice that G and k are uniquely determined by V). Thus, if $V \supset \varphi$ is valid in the system, then every run ρ' satisfying $v(G,\rho',k) = V$ must also satisfy φ. We now have:

Corollary 21: Let φ be a fact about the input and the existence of failures, and let $V = \hat{V}(\rho,\ell)$. Then $(\rho,\ell) \models C_{\mathcal{N}}\varphi$ iff $V \supset \varphi$ is valid in the system.

Proof: Let $V = \hat{V}(\rho,\ell)$. Suppose $V \supset \varphi$ is valid in the system. By Lemma 17, we have $\hat{V}(\rho,\ell) = \hat{V}(\rho',\ell)$ for all runs ρ' such that $(\rho,\ell) \sim (\rho',\ell)$, and hence that $(\rho',\ell) \models V$ for all such ρ'. Given that $V \supset \varphi$ is valid in the system, we have $(\rho',\ell) \models \varphi$ for all such ρ'. It follows that $(\rho,\ell) \models C_{\mathcal{N}}\varphi$.

For the other direction, suppose that $V \supset \varphi$ is not valid in the system. Since $V \supset \varphi$ is not valid in the system, let η be a run such that $(\eta,\ell) \models V$ and yet $(\eta,\ell) \not\models \varphi$. We will construct a run ζ such that $(\rho,\ell) \sim (\zeta,\ell)$, ζ and η have the same input, and ζ and η are the same with respect to the existence of failures (i.e., ζ will be failure-free iff η is). Since φ is a fact about the input and the existence of failures, $(\eta,\ell) \not\models \varphi$ will imply $(\zeta,\ell) \not\models \varphi$. Since, in addition, $(\rho,\ell) \sim (\zeta,\ell)$, we will have that $(\rho,\ell) \not\models C_{\mathcal{N}}\varphi$.

We construct ζ in two steps. We first construct a run ξ with the input of η satisfying $(\rho,\ell) \sim (\xi,\ell)$. Let ξ be the run with the failure pattern of ρ and the input of η. Given that ρ and ξ have the same failure pattern, and that $\hat{G}$ and $\hat{k}$ depend only on the failure pattern, we have that $\hat{G}(\rho,\ell) = \hat{G}(\xi,\ell)$ and $\hat{k}(\rho,\ell) = \hat{k}(\xi,\ell)$. Let us denote these values by $\hat{G}$ and $\hat{k}$. Since $(\eta,\ell) \models V$, we have $v(\hat{G},\rho,\hat{k}) = v(\hat{G},\eta,\hat{k})$, and hence $\mathcal{G}_{\hat{G}}(\rho,\hat{k}) = \mathcal{G}_{\hat{G}}(\eta,\hat{k})$. Since ξ and ρ have the same failure pattern, the *unlabeled* graphs underlying $\mathcal{G}_{\hat{G}}(\xi,\hat{k})$ and $\mathcal{G}_{\hat{G}}(\rho,\hat{k})$ (and hence also $\mathcal{G}_{\hat{G}}(\eta,\hat{k})$) are the same. Furthermore, since ξ and η have the same input, it follows that $\mathcal{G}_{\hat{G}}(\xi,\hat{k})$ and $\mathcal{G}_{\hat{G}}(\eta,\hat{k})$ (and hence also $\mathcal{G}_{\hat{G}}(\rho,\hat{k}))$ are equal. Since $\mathcal{G}_{\hat{G}}(\rho,\hat{k}) = \mathcal{G}_{\hat{G}}(\xi,\hat{k})$ implies $\hat{V}(\rho,\ell) = \hat{V}(\xi,\ell)$, we have $(\rho,\ell) \sim (\xi,\ell)$ by Lemma 19.

We now consider the existence of failures, and construct the desired run ζ. If there is a failure in η, then let ζ be a run differing from ξ only in that a processor fails after time ℓ in ζ. Clearly $(\xi,\ell) \sim (\zeta,\ell)$, and hence $(\rho,\ell) \sim (\zeta,\ell)$. Conversely, if η is failure-free, then let $\zeta = \eta$. Since η is failure-free, no processor in $\hat{G}$ knows of a failure at time $\hat{k}$ in η. Since processors in $\hat{G}$ have the same view at time $\hat{k}$ in both η and ρ, the same is true of ρ. It follows that $\hat{B} = B(\hat{G},\rho,\hat{k})$ is empty, and since $\hat{G} = P - \hat{B}$, we have that $\hat{G} = P$. Notice that ζ differs from ξ only in that processors in $\hat{G} = P$ do not fail in ζ, and hence that $(\xi,\ell) \sim (\zeta,\ell)$ by Lemma 18. Therefore, $(\rho,\ell) \sim (\zeta,\ell)$. In either case, $(\rho,\ell) \sim (\zeta,\ell)$, ζ and η have the same input, and are the same with respect to the existence of failures. It follows by the above discussion that $(\rho,\ell) \not\models C_{\mathcal{N}}\varphi$. □

Corollary 21 summarizes the sense in which the construction allows us to test whether relevant facts are common knowledge at a given point. Let us consider the computational complexity of performing such tests. The first step in applying Corollary 21 to determine whether a fact is common knowledge at (ρ,ℓ) is to construct $\hat{V}(\rho,\ell)$. Recall that a group of processors implicitly knows that a processor is faulty iff it knows of a message the processor failed to send. This is an easy fact to check given the communication graph corresponding to the group's view. It follows that computing every iteration of the construction can easily be done in polynomial time. Furthermore, since the construction is guaranteed to converge within $t+1$ iterations, it follows that $\hat{G}$ and $\hat{k}$, and hence also $\hat{V}$ can be computed locally in polynomial time (as long as $\hat{V}$ is of polynomial size). Recall that if φ is a practical fact, then it is possible to determine in polynomial time whether or not $V \supset \varphi$ is valid in the system. Thus, given a practical simultaneous choice problem $\mathcal{C}$, one polynomial-time implementation of a test for common knowledge of $\mathit{enabled}(a_i)$ is to construct the set $V = \hat{V}$ and determine whether $V \supset \mathit{enabled}(a_i)$ is valid in the system. As a result, Theorem 7 implies the following:

Theorem 22: If $\mathcal{C}$ is an implementable, practical simultaneous choice, then there is a polynomial-time optimal protocol for $\mathcal{C}$.

We reiterate the fact that the resulting protocol for $\mathcal{C}$ is optimal *in all runs*: actions are performed in runs of $\mathcal{F}_{\mathcal{C}}$ as soon as they could possibly be performed in runs of any other protocol, given the operating environment of the run. Thus, for example, simultaneous Byzantine agreement is performed in anywhere between 2 and $t+1$ rounds, depending on the pattern of failures (as is shown in [DM] to be the case in the crash failure model). Similarly, the firing squad problem can be performed in anywhere between 1 and $t+1$ rounds after a "start" signal is received. Paradoxically, in all these cases, the simultaneous actions can be performed quickly only when many failures become known to the nonfaulty processors. In particular, if there are no failures, no fact about the input is common knowledge less than $t+1$ rounds after it is first determined to hold.

Notice that every processor, faulty or nonfaulty, is able to compute the set $\hat{V}(\rho,\ell)$ locally. As a result, the following proposition shows that a fact is common knowledge to the nonfaulty processors iff it is common knowledge to all processors.

Proposition 23: Let φ be an arbitrary fact. In the omissions model, $C_{\mathcal{N}}\varphi \equiv C_P\varphi$ is valid in all systems running a full-information protocol.

Proof: By Theorem 2, it is enough to show that $(\rho,\ell) \overset{P}{\sim} (\rho',\ell)$ iff $(\rho,\ell) \overset{\mathcal{N}}{\sim} (\rho',\ell)$ for all runs ρ and ρ' and times ℓ. The 'if' direction is trivial, since $\mathcal{N} \subseteq P$. The proof of the other direction is identical to the proof of Lemma 17, interpreting $\sim$ as $\overset{P}{\sim}$. □

Proposition 23 implies that all processors (even the faulty processors) know exactly what actions are commonly known to be enabled in runs of $\mathcal{F}_{\mathcal{C}}$. Thus, in this model the protocol $\mathcal{F}_{\mathcal{C}}$ is guaranteed to satisfy a stronger version of simultaneous choice problems, in which condition (ii) is replaced by

(ii′) if a_i is performed by *any* processor (faulty or nonfaulty), then it is performed by *all* processors simultaneously.

Furthermore, since when an action is performed it is performed simultaneously by all processors, and since no other action is ever performed, there is no need for processors to continue sending messages after performing actions in runs of $\mathcal{F}_{\mathcal{C}}$ in this model. We can therefore further reduce the communication of $\mathcal{F}_{\mathcal{C}}$ by having processors halt after performing a simultaneous action. As a result, the following is an optimal protocol for any implementable simultaneous choice problem $\mathcal{C}$, an optimal protocol simpler than the protocol $\mathcal{F}_{\mathcal{C}}$:

```
repeat every round
    send current view to every processor
until $C_{\mathcal{N}}enabled(a_i)$ holds for some $a_i$;
$j \leftarrow \min\{i : C_{\mathcal{N}}enabled(a_i)$ holds$\}$;
perform $a_j$;
halt.
```

The fact that in the omissions model the information in $\hat{V}(\rho,\ell)$ is essentially all that is common knowledge at a given point has interesting implications about the type of simultaneous actions that can be performed in this model. For example, recall that in the traditional simultaneous Byzantine agreement or consensus problems (see [PSL], [F], [DM]), the processors are only required to decide, say, v in case they all start with an initial value of v. It would be more pleasing, however, if they could decide v whenever the majority of initial values are v. This is clearly impossible, since some processors may be silent throughout the run. However, consider a protocol for simultaneous Byzantine agreement which is similar to $\mathcal{F}_c$, except that when some *enabled*(a_i) becomes common knowledge (which happens exactly when $\hat{V}$ becomes non-empty), the processors choose the value that appears in the majority of the initial values recorded in $\hat{V}(\rho, \ell)$ as their decision value. In this case, the processors actually approximate majority fairly well: If more than $(n+t)/2$ of the initial values are v, then v will be chosen. In fact, we can show that the approximation is bad only in runs in which agreement is obtained early. In particular, if agreement cannot be obtained before time $t+1$ (this would happen in runs ρ for which $\hat{V}(\rho, \ell)$ contains only empty views for every $\ell \leq t$), then the value agreed upon would be the majority value in case more than $n/2+1$ of the processors have the same initial value. Furthermore, a protocol for *weak* (exact) majority *does* exist: A protocol that either decides that there was a failure or decides on the true majority value.

Since messages from faulty processors can convey new information about the failure pattern, such messages *do* affect the construction. Therefore, the behavior of faulty processors, even after they have been discovered to be faulty, plays an important role in determining what facts become common knowledge and when. In the crash failure model, however, a failed processor does not communicate with other processors after its failing round and has little impact on what facts become common knowledge. This is an essential property of the omissions model operationally distinguishing it from the crash failure model.

We note, however, that all of the analysis in this subsection applies to the crash failure model, with all of the proofs applying verbatim when restricted to the crash failure model. We thus have:

Proposition 24: In the crash failure model, $(\rho, \ell) \models C_{\mathcal{N}}\varphi$ iff it is the case that $(\rho', \ell) \models \varphi$ for all ρ' satisfying $\hat{V}(\rho, \ell) = \hat{V}(\rho', \ell)$.

Thus, the set $\hat{V}(\rho, \ell)$ completely characterizes what facts are common knowledge at the point (ρ, ℓ) in the crash failure model as well. Since the same proofs show that the construction characterizes the connected components of the similarity graph in both the omissions and the crash failure model, the similarity graph in the omissions model is simply an extension of the similarity graph in the crash failure model, maintaining the same connected components. This implies that in a run of the omission model having a failure pattern consistent with the crash failure model, exactly the same facts about the input and the existence of failures are common knowledge at any given time in both the crash failure and the omissions model. (However, as a result of the difference in the types of failures possible in the two failure models, different facts about the failure pattern are common knowledge at the corresponding points.) Ruben Michel has independently characterized the similarity graph in variants of the crash failure model (see [Mi]). For the crash failure model itself, he has an alternative construction that also characterizes the connected components of the similarity graph.

As in the omissions model, it follows from Proposition 24 that our construction can be used to derive efficient optimal protocols for simultaneous choice problems in the crash failure model, thus slightly extending [DM]. We therefore have the following:

Corollary 25: Let $\mathcal{C}$ be an implementable, practical simultaneous choice. In the crash failure model, there is a polynomial-time optimal protocol for $\mathcal{C}$.

As a final remark, let k_i and G_i be the intermediate results of beginning the construction at the point (ρ, ℓ), and denote $v(G_i, \rho, k_i)$ by V_i. Consider the operator $\mathcal{E}$ defined by $\mathcal{E}(V_i) = V_{i+1}$ for all i. We find it interesting that $\hat{V}$, which is the greatest fixpoint of the operator $\mathcal{E}$, characterizes the facts φ for which $C_{\mathcal{N}}\varphi$ holds, where we know from [HM] that $C_{\mathcal{N}}\varphi$ is the greatest fixpoint of $X \equiv E_{\mathcal{N}}(\varphi \wedge X)$.

6.2 Receiving Omissions

In the omissions model, faulty processors fail only to *send* messages. In this subsection, we consider the symmetric *receiving omissions* model, in which faulty processors fail only to *receive* messages. While at first glance these models seem quite similar, they are actually extremely different. In particular, we will see that testing for common knowledge in this model becomes trivial. As a result, there are simple, efficient optimal protocols for practical simultaneous choice problems in this model.

One intriguing difference between the omissions model and the receiving omissions model is the following. We have seen in the omissions model that in some cases a fact (for example, the arrival of a "start" signal) does not become common knowledge until as many as $t+1$ rounds after it is first determined to hold. Intuitively, the attainment of common knowledge is delayed by the possibility that a processor might fail to send a message determining that the fact holds. However, in the receiving omission model even *faulty* processors send all message required by the protocol. Since nonfaulty processors receive all messages sent to them, in runs of a full-information protocol all nonfaulty processors have a complete view of the first k rounds at time $k+1$. We can thus show the following:

Theorem 26: Let φ be a fact about the first k rounds. In the receiving omissions model, $(\rho, k) \models \varphi$ iff $(\rho, k+1) \models C_{\mathcal{N}}\varphi$.

The proof of this result depends on the notion of a fact being valid at time k: A fact φ is said to be *valid* (in the system) *at time* k if for all runs ρ we have $(\rho, k) \models \varphi$. We remark that the following variant of the induction rule holds:

If	$\varphi \supset E_{\mathcal{S}}\varphi$	is valid at time k,
then	$\varphi \supset C_{\mathcal{S}}\varphi$	is valid at time k.

Proof: Since φ is a fact about the first k rounds, $(\rho, k) \models \varphi$ iff $(\rho, k+1) \models \varphi$. Thus, it is enough to show that $(\rho, k+1) \models \varphi$ iff $(\rho, k+1) \models C_{\mathcal{N}}\varphi$. Clearly, $(\rho, k+1) \models C_{\mathcal{N}}\varphi$ implies $(\rho, k+1) \models \varphi$. Conversely, suppose $(\rho, k+1) \models \varphi$. During round $k+1$ in ρ every processor sends its entire view to all processors, so at time $k+1$ all nonfaulty processors have a complete view of the first k rounds of ρ. Since φ is a fact about the first k rounds, $(\rho, k+1) \models E_{\mathcal{N}}\varphi$. We have just shown that $\varphi \supset E_{\mathcal{N}}\varphi$ is valid at time $k+1$, so $\varphi \supset C_{\mathcal{N}}\varphi$ is valid at time $k+1$ as well. Thus, $(\rho, k+1) \models \varphi$ implies $(\rho, k+1) \models C_{\mathcal{N}}\varphi$. □

As a consequence of Theorem 26, polynomial-time optimal protocols for practical simultaneous choice problems are very simple in this model. Again, by polynomial-time here we will mean polynomial in n, t, and the round number ℓ.

Corollary 27: Let $\mathcal{C}$ be an implementable, practical simultaneous choice. In the receiving omissions model, there is a polynomial-time optimal protocol for $\mathcal{C}$.

Proof: Since $\mathcal{C}$ is implementable, Theorem 7 implies that $\mathcal{F}_{\mathcal{C}}$ is an optimal protocol for $\mathcal{C}$. It remains to show that $\mathcal{F}_{\mathcal{C}}$ can be implemented in polynomial time. Since the messages sent by $\mathcal{F}_{\mathcal{C}}$ can clearly be computed in polynomial time, we need only show how to implement the tests for common knowledge of the conditions $\mathit{enabled}(a_i)$ in polynomial time. We claim that $(\rho, \ell) \models C_{\mathcal{N}}\mathit{enabled}(a_i)$ iff $\mathcal{G}(\rho, \ell-1) \supset \mathit{enabled}(a_i)$ is valid in the system. Since $\mathcal{C}$ is a practical simultaneous choice problem, determining whether $\mathcal{G}(\rho, \ell-1) \supset \mathit{enabled}(a_i)$ is valid in the system can be done in polynomial time. As $\mathcal{G}(\rho, \ell-1)$ can be determined by all nonfaulty processors at (ρ, ℓ) in polynomial-time, this will yield a polynomial-time implementation of a test for common knowledge of $\mathit{enabled}(a_i)$, and we will be done. Suppose $\mathcal{G}(\rho, \ell-1) \supset \mathit{enabled}(a_i)$ is valid in the system. Theorem 26 implies that $\mathcal{G}(\rho, \ell-1)$ is common knowledge at (ρ, ℓ), and it follows that $(\rho, \ell) \models C_{\mathcal{N}}\mathit{enabled}(a_i)$. Conversely, suppose $(\rho, \ell) \models C_{\mathcal{N}}\mathit{enabled}(a_i)$. Let ζ be a run satisfying $\mathcal{G}(\rho, \ell-1)$. A proof similar to the base case of Lemma 11 shows that $(\rho, \ell) \sim (\zeta, \ell)$. Since $(\rho, \ell) \models C_{\mathcal{N}}\mathit{enabled}(a_i)$, it follows that $(\zeta, \ell) \models \mathit{enabled}(a_i)$. Thus, $\mathcal{G}(\rho, \ell-1) \supset \mathit{enabled}(a_i)$ is valid in the system, as desired. □

The results of this section point out a number of interesting differences between the omissions model and the receiving omissions model. For example, consider the distributed firing squad problem. First, Theorem 26 implies that all nonfaulty processors are able to fire in the receiving omission model exactly one round after the first "start" signal is received. Recall that in the omissions model, firing may delayed as many as $t+1$ rounds. Second, since a faulty processor p might fail to receive all messages, it is not possible to guarantee that p will ever fire following the receipt of a "start" signal by a nonfaulty processor. In the omissions model we have shown that it is possible to guarantee that *all* processors perform any action (e.g., "firing") performed by the nonfaulty processors. Finally, notice that faulty processors may sometimes be unable to halt, or terminate their participation in a distributed firing squad protocol, even long after the nonfaulty processors have fired: A processor p receiving no messages or "start" signals at all can never halt since at any point it is possible (according to p's view) that it will be the only processor in the system to receive a "start" signal. In this case, *optimal* protocols must require the nonfaulty processors to fire one round later, and hence p must be able to send this information to the nonfaulty processors. In contrast, in the omissions model it is possible to guarantee that all processors halt as soon as an action is performed in the system. These remarks show that while at first glance the assignment of responsibility for undelivered messages to sending or to receiving processors may seem arbitrary, the assignment has a dramatic effect on when facts become common knowledge, and hence on the behavior of optimal protocols. Since such a simple modification of the omissions model results in the collapse of the combinatorial structure underlying the model (witness Theorem 26), we consider this to be an indication that the omissions model is not a robust model of failure.

6.3 Generalized Omissions

We have just seen that the choice of whether sending or receiving processors are responsible for undelivered messages has a dramatic effect on the structure of the omissions model. Perhaps a more natural model of failure is the *generalized omissions* model, in which a faulty processor may fail both to send *and* to receive messages. This section is concerned with the design of optimal protocols for simultaneous choice problems in this model. We have seen that Theorem 7 implies the protocol $\mathcal{F}_c$ is an optimal protocol in this model, and that Theorem 10 implies this protocol can be implemented in polynomial-space. As in previous sections, the remaining question is whether there are efficient optimal protocols in this model. The principal result of this section is that testing for common knowledge in the generalized omissions model in NP-hard. Using the close relationship between common knowledge and simultaneous actions, we obtain as a corollary that optimal protocols for most any simultaneous choice problem in this model require processors to perform NP-hard computations. Consequently, for example, in this model there can be no efficient optimal protocol for simultaneous Byzantine agreement or the distributed firing squad problem. This is a dramatic difference between the generalized omissions model and the more benign failure models, where, as we have seen, efficient optimal protocols *do* exist.

One important difference between the generalized omissions model and simpler variants of the omissions model is that in the generalized omissions model undelivered messages do not necessarily identify the set of faulty processors, but merely place constraints on their possible identities: Either the sender or the intended receiver of every undelivered message must be faulty. The faulty processors must therefore induce a "vertex cover" of the undelivered messages. Recall that in our analysis of the omissions failure model, determining the number and the identity of the faulty processors given the labeled communication graph of a point played a crucial role in characterizing the facts that are common knowledge at a point. In that model, a processor is known to be faulty iff it is known that a message it was supposed to send was not delivered, a fact easily determined from the labeled communication graph. In the generalized omissions model, however, even determining the number (and not necessarily the identities) of processors implicitly known to be faulty essentially involves computing the size of the minimal vertex cover of a graph, a problem known to be NP-complete (see [GJ]). It is with this intuition that we now proceed to show that determining whether certain facts are common knowledge is computationally prohibitive in the generalized omissions model, assuming P$\neq$NP.

However, in order to study the complexity of testing for common knowledge in the generalized omissions model in a meaningful way, we are once again faced with the need to restrict our attention to a class of facts that includes all of the facts that may arise in natural simultaneous choice problems, and excludes anomalous cases. For example, if φ is valid in the system, then so is $C_{\mathcal{N}}\varphi$, and testing whether φ is common knowledge is a trivial task. On the other hand, one can imagine facts involving excessive computational complexity of a type irrelevant to simultaneous choice problems. Consider, for instance, a fact φ with the property that the communication graph of any point satisfying φ encodes information allowing the solution of all problems in NP of size smaller than the number of processors in the system. Whereas it seems unlikely that such a fact exists, this fact is probably very hard to prove, and is definitely not the business of this paper to do. We are therefore led to make the following restriction. A fact φ is said to be *admissible* within a class of systems running a full-information protocol if (i) for all systems within this class neither φ nor $\neg\varphi$ is valid in the system, and (ii) there is a polynomial-time algorithm explicitly constructing for each system a labeled communication graph $\mathcal{G}(\rho, \ell)$ of minimal length having the property that $\mathcal{G}(\rho, \ell) \supset \varphi$ is valid in the system. We say that a simultaneous choice problem $\mathcal{C}$ is admissible if each condition *enabled*(a_i) is admissible within the class of systems determined by a full-information protocol and $\mathcal{C}$. We claim that any natural simultaneous choice is admissible. We can now state the fundamental result of this section which says, loosely speaking, that testing for common knowledge of admissible facts $\varphi_1, \ldots, \varphi_s$ is NP-hard.

Lemma 28: Let $\varphi_1, \ldots, \varphi_s$ be admissible, practical facts within a class of systems running a full-information protocol in the generalized omissions model. Given the graph $\mathcal{G}(\rho, \ell)$ of a point in such a system with $n > 2t$, the problem of determining whether $(\rho, \ell) \models \bigvee_i C_{\mathcal{N}}\varphi_i$ is NP-hard (in n).

The proof of Lemma 28 will follow shortly. Notice, however, that t is variable in the statement of this lemma, and in general may be $O(n)$. The proof of this result will not apply for a fixed t, nor to cases in which t is restricted, say, to be $O(\log n)$. In any case, it will follow that any standard implementation of our optimal knowledge-based protocols must be computationally intractable, unless P=NP. It is natural to ask whether this inefficiency is merely the result of having programmed our protocols using tests for common knowledge. It is conceivable, for instance, that there are optimal protocols for admissible simultaneous choice problems in the generalized omissions model that *are* computationally efficient. Intuitively, however, in order to perform a simultaneous action, an optimal protocol $\mathcal{P}$ must essentially determine whether any of the conditions *enabled*(a_i) is common knowledge. Corollary 6 implies that such a condition becomes common knowledge during the corresponding run of a full-information protocol as soon as it does during a run of $\mathcal{P}$. Thus, $\mathcal{P}$ must essentially determine whether such a fact is common knowledge during the corresponding run of a full-information protocol $\mathcal{F}$. Since Lemma 28 implies that this problem is NP-hard, computing the function $\mathcal{P}$ must be NP-hard as well. We now make this argument precise.

Recall that a protocol is formally a function mapping n, t, and a processor's view to a list of the actions the processor should perform, followed by a list of the messages it is required to send in the following round. We say that a protocol is *communication-efficient* if in a system of n processors the size of the messages each processor is required to send during round ℓ is polynomial in n and ℓ. In the following result we show that the problem of computing the function corresponding to a communication-efficient optimal protocol for a simultaneous choice problem is NP-hard. Hence, no such protocol can be computationally efficient.

Theorem 29: Let $\mathcal{P}$ be a communication-efficient, optimal protocol for an admissible, practical simultaneous choice $\mathcal{C}$. The problem of computing (the function) $\mathcal{P}$ is NP-hard (in n).

Proof: Notice that since $\mathcal{P}$ is a protocol for $\mathcal{C}$, the problem $\mathcal{C}$ must be implementable, and Theorem 7 implies that the full-information protocol $\mathcal{F}_\mathcal{C}$ must be an optimal protocol for $\mathcal{C}$. Let $\Sigma = \{\Sigma(n,t) : n \geq t+2\}$ be the class of systems determined by $\mathcal{C}$ and $\mathcal{F}_\mathcal{C}$. Since $\mathcal{C}$ is an admissible, practical simultaneous choice, each condition *enabled*(a_i) must be an admissible, practical fact within Σ. By Lemma 28, given the graph $\mathcal{G}(\rho,\ell)$ of a point (ρ,ℓ) in a system $\Sigma(n,t)$ with $n > 2t$, the problem of determining whether $(\rho,\ell) \models \bigvee_i C_\mathcal{N}\, \mathit{enabled}(a_i)$ is NP-hard. We will exhibit a Turing reduction from this problem to the problem of computing $\mathcal{P}$; that is, given the graph $\mathcal{G}(\rho,\ell)$ of a point (ρ,ℓ) in a system $\Sigma(n,t)$ with $n > 2t$, we will show how to use $\mathcal{P}$ to determine in polynomial time whether $(\rho,\ell) \models \bigvee_i C_\mathcal{N}\, \mathit{enabled}(a_i)$. Having exhibited such a reduction, we will have shown that the problem of computing $\mathcal{P}$ is NP-hard.

Let ρ be a run of $\mathcal{F}_\mathcal{C}$ in a system $\Sigma(n,t)$ with $n > 2t$, and let ζ be the corresponding run of $\mathcal{P}$. It follows from the definition of $\mathcal{F}_\mathcal{C}$ that $(\rho,\ell) \models \bigvee_i C_\mathcal{N}\, \mathit{enabled}(a_i)$ iff the nonfaulty processors perform a simultaneous action no later than time ℓ in ρ. Since $\mathcal{F}_\mathcal{C}$ and $\mathcal{P}$ are both optimal protocols for $\mathcal{C}$, the nonfaulty processors perform simultaneous actions at the same times during ρ and ζ. Since $n > 2t$, there must be at least $t+1$ nonfaulty processors in both runs, so the nonfaulty processors simultaneously perform an action no later than time ℓ in either run iff $t+1$ processors do so. Therefore, $(\rho,\ell) \models \bigvee_i C_\mathcal{N}\, \mathit{enabled}(a_i)$ iff $t+1$ processors perform a simultaneous action no later than time ℓ in ζ.

One algorithm for determining whether $t+1$ processors *do* perform a simultaneous action no later than time ℓ in ζ is to construct the view of each processor in ζ at each time k before time ℓ, and use $\mathcal{P}$ to determine when processors are required to perform actions. Suppose we have constructed the view of each processor at time $k-1$ in ζ; let us consider the problem of constructing the view of a processor p at time k. Processor p's view at (ζ, k) consists of p's name, the time k, a list of the messages received by p during the first k rounds of ζ, and a list of the input received by p during the first k rounds of ζ. Recall that since ρ is a run of full-information protocol, the graph $\mathcal{G}(\rho, \ell)$ is actually an encoding of the operating environment during the first ℓ rounds of ρ, and hence also of ζ. Given the views of all processors at time $k-1$, the protocol $\mathcal{P}$ determines what message each processor is required to send to p, and $\mathcal{G}(\rho, \ell)$ determines which of these messages are actually delivered to p in ζ. Since $\mathcal{P}$ is communication-efficient, each of these messages is of size polynomial in n and k. Furthermore, the input received by p during round k labels the node $\langle p, k\rangle$ of $\mathcal{G}(\rho, \ell)$. Since $\mathcal{C}$ is practical, this input is of constant size. Thus, given each processor's view at time $k-1$, we can use the graph $\mathcal{G}(\rho, \ell)$ and an oracle for $\mathcal{P}$ to construct the view of each processor at time k of ζ in polynomial time. (An oracle for $\mathcal{P}$ is an oracle that, given the view of a processor p at a point (ρ, ℓ), in one step determines what actions $\mathcal{P}$ requires p to perform at time ℓ, and constructs the messages $\mathcal{P}$ requires p to send during round $\ell+1$.)

Consider the following algorithm:

```
action_performed ← false;
$k$ ← 0;
repeat
  for all processors $p$ do
    determine whether $\mathcal{P}$ requires $p$ to perform any action at time $k$, and
    construct the messages $\mathcal{P}$ requires $p$ to send during round $k+1$;
  endfor
  if $t+1$ processors perform actions at time $k$
    then action_performed ← true;
  $k \leftarrow k+1$;
until $k > \ell$ or action_performed;
if action_performed
  then halt with "yes"
  else halt with "no".
```

From the previous discussion it is clear that given any oracle for $\mathcal{P}$, this algorithm determines in polynomial time whether $t+1$ processors perform actions simultaneously no later than time ℓ in ζ, and hence whether $(\rho, \ell) \models \bigvee_i C_{\mathcal{N}}\, enabled(a_i)$. □

As an immediate corollary of Theorem 29, we have the following:

Corollary 30: Let $\mathcal{C}$ be an admissible practical simultaneous choice problem. If there is a polynomial-time optimal protocol for $\mathcal{C}$, then P=NP.

Corollary 30 implies that optimal protocols for simultaneous choice problems as simple as the distributed firing squad problem or simultaneous Byzantine agreement are computationally infeasible in the generalized omissions model, assuming P $\neq$ NP. In fact, we do not know whether these problems can be implemented in polynomial time even using an NP oracle. The best we can do in the generalized omissions model is implement them using polynomial-space computations, as in the proof of Theorem 10. We consider the question of determining the exact complexity of implementing admissible practical simultaneous choice problems in this model an interesting open problem.

We now proceed to prove Lemma 28. First, however, we state a result that will be very useful in the proof of Lemma 28. Roughly speaking, it says that if a group of processors can (jointly) prove that they are nonfaulty, then their views become common knowledge at the end of the following round.

Lemma 31: Let S be a set of processors and let $\overline{S} = P - S$. Let ρ be a run of a full-information protocol. If the processors in S implicitly know at $(\rho, \ell - 1)$ that $\overline{S}$ contains t faulty processors, then the joint view of S at $(\rho, \ell - 1)$ is common knowledge at (ρ, ℓ).

Proof: Let φ = "V is the joint view of S at time $\ell - 1$", where $V = v(S, \rho, \ell - 1)$. Suppose $(\rho', \ell) \models \varphi$. Given that S has the same joint view at $(\rho, \ell - 1)$ and at $(\rho', \ell - 1)$, and since S implicitly knows at $(\rho, \ell - 1)$ that $\overline{S}$ contains t faulty processors, S implicitly knows the same at $(\rho', \ell - 1)$. In particular, the processors in S must be nonfaulty in ρ', and each must successfully send its view to all processors during round ℓ of ρ'. Since all nonfaulty processors will receive these messages, we have $(\rho', \ell) \models E_{\mathcal{N}}\varphi$. It follows that $\varphi \supset E_{\mathcal{N}}\varphi$ is valid at time ℓ, and the induction rule implies $\varphi \supset C_{\mathcal{N}}\varphi$ is valid at time ℓ as well. Thus, $(\rho, \ell) \models \varphi$ implies $(\rho, \ell) \models C_{\mathcal{N}}\varphi$. □

(We note in passing that a converse to Lemma 31 is also true: If the joint view at time $\ell - 1$ of a set S of processors is common knowledge at time ℓ, then the processors in some set $S' \supseteq S$ must implicitly know at time $\ell - 1$ that there are t faulty processors among the members of $\overline{S}'$.)

In addition to Lemma 31, the following result, analogous to Lemma 11 in the omissions model, will be of use in the proof of Lemma 28.

Lemma 32: Let ρ and ρ' be runs differing only in the (faulty) behavior displayed by processor p after time k, and suppose no more that f processors fail in either ρ or ρ'. If $\ell - k \leq t + 1 - f$, then $(\rho, \ell) \sim (\rho', \ell)$.

Proof: The proof is analogous to the proof of Lemma 11, with the added observation that if p sends no messages after (an arbitrary) time k in ζ, then $(\zeta, \ell) \sim (\zeta', \ell)$ where ζ' differs from ζ in that p receives messages from an arbitrary set of processors during round k. □

Finally, as previously mentioned, the proof of Lemma 28 involves a reduction from the *Vertex Cover* problem. This is the problem (see [GJ]) of determining, given a graph $G = (V, E)$ and a positive integer M, whether G has a *vertex cover* of size M or less; that is, a subset $\mathcal{V} \subseteq V$ such that $|\mathcal{V}| \leq M$ and, for each edge $\{v, w\} \in E$, at least one of v or w belongs to $\mathcal{V}$.

Theorem [Karp]: *Vertex Cover* is NP-complete.

We now prove Lemma 28.

Proof of Lemma 28: We will exhibit a Turing reduction from Vertex Cover to the problem of testing for common knowledge of $\varphi_1, \ldots, \varphi_s$, and it will follow that this problem is NP-hard.

Since every graph $G = (V, E)$ is $|V|$-coverable, the following is an algorithm for *Vertex Cover*:

```
$m \leftarrow |V|$;
while $G$ has a vertex cover of size $m - 1$ do
    $m \leftarrow m - 1$;
if $m \leq M$
    then return "G has a vertex cover of size M"
    else return "G has no vertex cover of size M".
```

To implement this test, it is enough to implement a test that, given an m-coverable graph G, determines whether G is $(m-1)$-coverable. Every graph $G = (V, E)$ clearly has a vertex cover of size $|V| - 1$. In addition, it is possible to determine whether G has a vertex cover of size $|V| - 2$ in polynomial time. Similarly, it is easy to determine whether G has a vertex cover of size 0 in polynomial time. We show that if $1 \leq m \leq |V| - 2$ and G is m-coverable, then it is possible to construct in polynomial time a graph $\mathcal{G}(\rho, \ell)$ with the property that $(\rho, \ell) \models \bigvee_i C_{\mathcal{N}} \varphi_i$ iff G is *not* $(m-1)$-coverable. The point (ρ, ℓ) will be a point of a system $\Sigma(n, t)$ with $n > 2t$ from the class under consideration (i.e., the class of systems running a full-information protocol in the generalized omissions model). Thus, given an oracle for testing for common knowledge of $\varphi_1, \ldots, \varphi_s$, we will have a polynomial-time test for the $(m-1)$-coverability of G. It will follow that testing for common knowledge of $\varphi_1, \ldots, \varphi_s$ is NP-hard.

Fix a graph $G = (V, E)$ and an integer m satisfying $1 \leq m \leq |V| - 2$. Let $n = |V| + m + 3$ and $t = m + 2$, and let $\Sigma(n, t)$ be a system from the class under consideration.

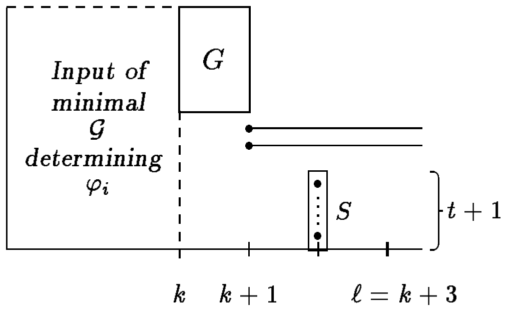


Figure 5: Embedding a graph G in a run ρ.

Notice that since $|V| \geq m+2$, we have $n > 2t$. Since each fact φ_i is admissible, we can explicitly construct in polynomial time a labeled communication graph (of a point in $\Sigma(n,t)$) of minimal length determining φ_i. Of these graphs, let $\mathcal{G}$ be one of minimal length, say of length k. Let ρ be a run of $\Sigma(n,t)$, illustrated in Figure 5, satisfying the following conditions: (i) the input received in the first k rounds of ρ is the same as in $\mathcal{G}$, and no input is received after time k; (ii) all messages in the first k rounds are delivered; (iii) in round $k+1$, the only undelivered messages are as follows: no message is delivered from processor p_v to p_w in round $k+1$ of ρ iff there is an edge from v to w in G (that is, the graph G is represented by the undelivered messages during round $k+1$); (iv) two additional processors f_1 and f_2 are silent from time $k+1$ in ρ, and all other messages after time $k+1$ are delivered; and (v) a set S of $t+1$ additional processors do not fail in ρ. Since G has a vertex cover $\mathcal{V}$ of size m, one failure pattern consistent with the undelivered messages in ρ is that p_v is faulty for every $v \in \mathcal{V}$ (accounting for the undelivered messages during round $k+1$ of ρ) and that both f_1 and f_2 are faulty. Given that $t = m+2$ processors fail in this failure pattern, there is a run ρ of $\Sigma(n,t)$ satisfying the required conditions. Since the graph $\mathcal{G}$ determining the input of $\mathcal{G}(\rho,k)$ can be constructed in polynomial time, setting $\ell = k+3$, the graph $\mathcal{G}(\rho,\ell)$ can be constructed in polynomial time as well. It remains to show that $(\rho,\ell) \models \bigvee_i C_{\mathcal{N}}\varphi_i$ iff G is not $(m-1)$-coverable.

Suppose G has no vertex cover of size $m-1$, and let F be the set of processors failing in ρ. Since f_1 and f_2 must be faulty (each fails to the $t+1$ processors in S), $F' \overset{\text{def}}{=} F - \{f_1, f_2\}$ must account for every undelivered message during round $k+1$. If there is an edge from v to w in G, then no message from p_v to p_w is delivered in round $k+1$, and one of p_v or p_w must be in F'. It follows that F' must induce a vertex cover of G. Since G has no vertex cover of size $m-1$, F' must contain at least m processors, and F at least $t = m+2$. Thus, the processors in S implicitly know at time $k+2$ that their complement $\overline{S} = P - S$ contains t faulty processors. By Lemma 31, their views at time $k+2$ must be common knowledge at time $k+3$. These views contain a complete description of $\mathcal{G}(\rho,k)$, and hence the identity of $\mathcal{G}(\rho,k)$ is common knowledge at (ρ, ℓ). Recall that $\mathcal{G}$ was chosen to be a graph determining φ_i for some i. If $\mathcal{G}$ does not specify a failure, then $\mathcal{G}(\rho, k) = \mathcal{G}$, and it follows that $(\rho, \ell) \models C_{\mathcal{N}}\varphi_i$. On the other hand, if $\mathcal{G}$ *does* specify a failure, then φ_i is determined by the input to the first k rounds of $\mathcal{G}$ and the existence of a failure. Notice that the failure of f_1 and f_2 is also recorded in the view of S at time $k+2$, and hence is also common knowledge at (ρ, ℓ). Thus, both the input to the first k rounds of $\mathcal{G}$ and the existence of a failure are common knowledge at time ℓ, and it follows that $(\rho, \ell) \models C_{\mathcal{N}}\varphi_i$. In either case, we have $(\rho, \ell) \models \bigvee_i C_{\mathcal{N}}\varphi_i$.

Conversely, suppose G *does* have a vertex cover of size $m - 1$. Without loss of generality, at most $t - 1$ processors fail in ρ. First, we claim that $(\rho, \ell) \sim (\zeta, \ell)$ where ζ is a failure-free run with the input of ρ. Since f_1 and f_2 fail only after time $k + 1 = \ell - 2$, two applications of Lemma 32 imply that $(\rho, \ell) \sim (\rho', \ell)$ where ρ' differs from ρ in that f_1 and f_2 do not fail in ρ'. Since at most $t - 3$ processors fail in ρ' and $k = \ell - 3$, by Lemma 32 we have $(\rho', \ell) \sim (\zeta, \ell)$. Second, we claim that for each φ_i there is a run η_i not satisfying φ_i that differs from $\mathcal{G}$ only after time $k - 1$. If $k = 0$, then since φ_i is admissible and hence not valid in the system, such a run must certainly exist. On the other hand, if $k > 0$, then since $\mathcal{G}$ was chosen to be a labeled communication graph of minimal length determining φ_j for some φ_j, such a run must exist in this case as well. Now, let η'_i be a run having the input of η_i, in which no processor fails before time ℓ, and in which processors become silent after time ℓ iff there is a failure in η_i. Since φ_i is a fact about the input and existence of failures, and since η_i does not satisfy φ_i, neither does η'_i. Let $\hat{\zeta}$ and $\hat{\eta}'_i$ be runs of $\mathcal{F}$ in the omissions model having the operating environments of ζ and η'_i, respectively. (Notice that these operating environments actually are operating environments of the omissions model.) Notice that no processor fails before time ℓ in either $\hat{\zeta}$ or $\hat{\eta}'_i$. It follows that $\hat{G}(\hat{\zeta}, \ell) = \hat{G}(\hat{\eta}'_i, \ell)$, and that $\hat{k}(\hat{\zeta}, \ell) = \hat{k}(\hat{\eta}'_i, \ell)$. We denote these values by $\hat{G}$ and $\hat{k}$, respectively. Since $t = m + 2$ and $m \geq 1$, we have that $t \geq 3$. Thus, $\hat{k} = \ell - (t + 1) \leq \ell - 4 = k - 1$. Recall that $\hat{\zeta}$ and $\hat{\eta}'_i$ have the same input (and no failures) through time $k - 1$. It follows that $\hat{V}(\hat{\zeta}, \ell) = \hat{V}(\hat{\eta}'_i, \ell)$. It follows by Lemma 19 that $(\hat{\zeta}, \ell) \sim (\hat{\eta}'_i, \ell)$ in the omissions model, and hence that $(\zeta, \ell) \sim (\eta'_i, \ell)$ in the generalized omissions model as well. Since $(\rho, \ell) \sim (\zeta, \ell)$, it follows that for each φ_i we have $(\rho, \ell) \sim (\eta'_i, \ell)$ and $(\eta'_i, \ell) \not\models \varphi_i$. Therefore, for each φ_i we have $(\rho, \ell) \not\models C_{\mathcal{N}}\varphi_i$, and hence $(\rho, \ell) \not\models \bigvee_i C_{\mathcal{N}}\varphi_i$. □

We have seen that, as a result of the uncertainty about the failure pattern, the complexity of determining whether admissible facts are common knowledge is dramatically greater in this model than in more benign models. It is conceivable, however, that this gap in complexity is due to the fact that faulty processors may fail both to send and to receive messages, and not merely due to the uncertainty about the failure pattern. We can show, however, that it is *precisely* due to this uncertainty that we observe this complexity gap. Consider the closely related failure model we have termed *generalized omissions with information*, a model differing from the generalized omissions model in that a processor not receiving a message can determine whether it or the sender is at fault. We can show that the construction used in the omissions model can also be used in this model to yield a set of views $\hat{V}(\rho, \ell)$ completely characterizing what facts are common knowledge at the point (ρ, ℓ).

Proposition 33: In generalized omissions with information, we have $(\rho, \ell) \models C_{\mathcal{N}}\varphi$ iff $(\rho', \ell) \models \varphi$ for all ρ' satisfying $\hat{V}(\rho', \ell) = \hat{V}(\rho, \ell)$.

All of the proofs in the omissions model hold when generalized to this model, with the exception that the construction must be started with a nonfaulty processor. (In particular, Lemma 16 holds only when the processors p and q are processors that do not fail to receive messages.) This exception says that faulty processors may not be able to perform all actions performed by the nonfaulty processors, but this is no surprise since the same is true in the receiving omissions model. Furthermore, the computation of the sets B_i in the construction now depends not only on the undelivered messages, but also on the additional information that receiving processors obtain regarding blame for the undelivered messages. As in the omissions model, this construction yields a method of deriving efficient tests for common knowledge of certain facts. Thus, it is again possible to design efficient optimal protocols:

Theorem 34: Let $\mathcal{C}$ be an implementable practical simultaneous choice. In generalized omissions with information, there is a polynomial-time optimal protocol for $\mathcal{C}$.

This shows that it is precisely the uncertainty about the failure pattern that is responsible for the observed gap in complexity, and not merely the fact that faulty processors may fail both by failing to send *and* to receive messages.

The uncertainty about the failure pattern in the generalized omissions model adds a new combinatorial structure to the similarity graph in this model that does not exist in other variants of the omissions model. Since it is possible to assign failure to processors in a number of different ways consistent with a pattern of undelivered messages, it is possible to play "pebbling games" with the failure pattern when constructing paths in the similarity graph, showing that one point is similar to another point by alternatively assigning responsibility for undelivered messages to the sender and to the receiver. In fact, in addition to increasing the difficulty of determining *whether* a fact is common knowledge at a point, this new combinatorial structure has interesting effects on *when* facts become common knowledge. Recall from the discussion at the end of Section 6.1 that the similarity graph in the omissions model is simply an extension of the similarity graph in the crash failure model, two points with crash failure patterns being similar in the crash failure model iff they are in the omissions model. As a result, our optimal protocol $\mathcal{F}_{\mathcal{C}}$ in the omissions model is also an optimal protocol when restricted to runs of the crash failure model. In the generalized omissions model, however, the similarity graph is *not* merely an elaboration of the similarity graph in the omissions model: A connected component in the similarity graph of the generalized omissions model may consists of several distinct connected components from the omissions model. As a result, optimal protocols in the generalized omissions model are not necessarily optimal when restricted to runs of the omissions model, as the following theorem shows is the case for simultaneous Byzantine agreement.

Theorem 35: No optimal protocol for simultaneous Byzantine agreement in the generalized omissions model is optimal when restricted to runs of the omissions model.

Proof: Let π be the failure pattern (involving at least $2t$ processors) in which processor p_i fails to send to processor p_{t+i} in round 1 (for $i = 1, \ldots, t$) and no other failures occur. Notice that π is a failure pattern of both the omissions model and the generalized omissions model. Let ρ be a run of a full-information protocol with the failure pattern π. We claim that some nonvalid fact about the initial configuration (in fact, the entire initial configuration) must be common knowledge at $(\rho, 2)$ in the omissions model; and that no nonvalid fact about the initial configuration is common knowledge at $(\rho, 2)$ in the generalized omissions model, from which it follows by Corollary 6 that no nonvalid fact about the initial configuration is common knowledge at time 2 in any run with failure pattern π of a protocol in the generalized omissions model. In the first case, any optimal protocol for simultaneous Byzantine agreement in the omissions model (the protocol $\mathcal{F}_c$, for example) halts at time 2. In the second case, Lemma 4 implies that *no* protocol for simultaneous Byzantine agreement in the generalized omissions model can halt at time 2. Therefore, no optimal protocol for simultaneous Byzantine agreement in the generalized omissions model is optimal when restricted to runs of the omissions model.

To see that some nonvalid fact about the initial configuration becomes common knowledge at $(\rho, 2)$ in the omissions model, notice that the set $\hat{V}(\rho, 2)$ is nonempty. The result follows by Corollary 21.

To see that no nonvalid fact about the initial configuration becomes common knowledge at $(\rho, 2)$ in the generalized omissions model, it is enough to show that $(\rho, 2) \sim (\zeta, 2)$ for all failure-free runs ζ. Shifting "pebbles," notice that $(\rho, 2) \sim (\rho', 2)$ where ρ' differs from ρ only in that processor p_1 is nonfaulty in ρ' and it is processor p_{t+1} that fails to receive the undelivered message from p_1 to p_{t+1} in round 1. Lemma 32 implies that $(\rho', 2) \sim (\rho'', 2)$ where ρ'' differs from ρ' only in that processor p_{t+1} does not fail to receive the message from processor p_1 in round one. Repeated applications of Lemma 32 now show that $(\rho'', 2) \sim (\eta, 2)$ where η is the failure-free run with the input of ρ''. Further applications of Lemma 32 to silence one by one each processor p_i, change its input, and revive p_i shows that $(\eta, 2) \sim (\zeta, 2)$ for all failure-free runs ζ. Therefore, $(\rho, 2) \sim (\zeta, 2)$ for all failure-free runs ζ, as desired. □

We remark that, for most simultaneous choice problems, the counterexample given in the proof of Theorem 35 can be used to show that no optimal protocol for this problem in the generalized omissions model is optimal when restricted to runs of the omissions model.

The results of this section indicate that the generalized omissions model seems to be a natural failure model that already displays some of the complex behavior of the more malicious models. We believe that this model is therefore a natural candidate for further study as an intermediate model on the way to understanding the mysteries of fault tolerance in truly malicious failure models.

7 Conclusions

This paper applies the theory of knowledge in distributed systems to the design and analysis of fault-tolerant protocols for a large and interesting class of problems. This is a good example of the power of applying reasoning about knowledge to obtain general, unifying results and a high-level perspective on issues in the study of unreliable systems. We believe that reasoning about knowledge will continue to be an effective tool in studying the basic structure and the fundamental phenomena in a large variety of problems in distributed computing.

Given the effectiveness of a knowledge-based analysis in the case of simultaneous actions (see also [DM]), it would be interesting to know whether such an analysis can shed similar light on the case of *eventually* coordinated actions. Dolev, Reischuk, and Strong show that the problem of performing eventually coordinated actions in such synchronous systems is quite different from that of performing simultaneous actions (see [DRS]). In addition to common knowledge, an analysis of eventually coordinated actions may be able to make good use of the notion of *eventual common knowledge* ([HM], [Mo]). We note that it is possible to show that for eventual choice problems there do not, in general, exist protocols that are *optimal in all runs.* For example, one can give two protocols for (eventual) Byzantine agreement with the property that for every operating environment one of these protocols will reach Byzantine agreement (i.e., all processors will decide on a value) by time 2 at the latest. However, if $t > 1$, it is well known that no single protocol can guarantee that agreement will be reached by time 2 in all runs. What is the best notion of optimality that *can* be achieved in eventual coordination?

We provide a method of deriving an optimal protocol for any given *implementable* specification of a simultaneous choice problem. However, in this work, we have completely sidestepped the interesting question of characterizing the problems that are and are not implementable in different failure models. We believe that a general analysis of the implementability of problems involving coordinated actions in different failure models will expose many of the important operational differences between the models. As an example, our specification of the distributed firing squad problem in the introduction is implementable in the variants of the omissions model, but *is not* implementable in more malevolent models, in which a faulty processor can falsely claim to have received a "start" message and otherwise seem to behave correctly (see [BL] and [CDDS] for definitions of versions of the firing squad problem that *are* implementable in the more malicious models).

In the generalized omissions model, we have shown how to derive optimal protocols for nontrivial simultaneous choice problems, requiring processors to perform polynomial-space computations between consecutive rounds. We have also shown an NP-hard lower bound for any communication-efficient protocol for such a problem that is optimal in all runs. Determining the precise complexity of this task is a nontrivial open problem, due to the interesting combinatorial structure underlying the generalized omissions model. It would also be interesting to extend our study to more malicious failure models, such as the *Byzantine* and the *authenticated Byzantine* models (see [F]). It is not immediately clear whether the notion of a failure pattern can be defined in these models in a protocol-independent fashion. Thus, it is not clear that the notion of optimality in all runs is well defined in such models. If such definitions are possible, we believe that the NP-hardness result from the generalized omissions model should extend to these models. (Our proof does show that testing for common knowledge in runs of the full-information protocol $\mathcal{F}$ in both models is NP-hard.) Capturing the precise combinatorial structure of the similarity graph in these models is bound to expose many of the mysterious properties of the models. We believe that this is an important first step in understanding these models.

As we have seen, there are no computationally-efficient optimal protocols for simultaneous choice problems in the generalized omissions model. Since it is unreasonable to expect processors to perform NP-hard computations between consecutive rounds of communication, it is natural to ask what is the earliest time at which such actions can be performed by *resource bounded* processors (e.g., processors that can perform only polynomial-time computations). Are there always guaranteed to be optimal protocols for such processors? How can they be derived? The analysis of this question is no longer as closely related to the question of when facts about the run become common knowledge. It seems that the information-based definition of knowledge that we presented in Section 3, used in many other papers as the definition of knowledge in a distributed system (see [CM], [DM], [FI], [HM], and [PR]), is not appropriate for reasoning about such questions. A major challenge motivated by this is the elaboration of the definition of knowledge presented in Section 3 to include notions of resource-bounded knowledge that would provide us with appropriate tools for analyzing such questions. Such a theory would provide notions such as *polynomial-time knowledge* and *polynomial-time common knowledge*, which would correspond to the actions and the simultaneous actions that polynomial-time processors can perform. Note that the fact that (suboptimal) polynomial-time protocols for the simultaneous Byzantine agreement problem exist even in the more malicious failure models imply that, given the right notions, many relevant facts should become polynomial-time common knowledge. Much work is left to be done.

Acknowledgments

We are greatly indebted to Oded Goldreich for his generous collaboration on a lemma used in a previous version of the proof of Lemma 28. We are also indebted to Joe Halpern for a thorough reading of a previous version of this paper. We thank Brian Coan, Oded Goldreich, Amos Israeli, Nancy Lynch, Moshe Vardi, and Jennifer Welch for comments that improved the presentation of this work. We also thank Paul Beame, Cynthia Dwork, Alan Fekete, Vassos Hadzilacos, Michael Merritt, Albert Meyer, David Peleg, and Larry Stockmeyer for stimulating discussions on the topic of this paper. This research was performed while both authors were at MIT. The work of the first author was primarily supported by an IBM Post-doctoral fellowship. The work of both authors was supported by the Office of Naval Research under contract N00014-85-K-0168, by the Office of Army Research under contract DAAG29-84-K-0058, by the National Science Foundation under grant DCR-8302391, and by the Defense Advanced Research Projects agency (DARPA) under contract N00014-83-K-0125.

References

[BL] J. Burns and N. A. Lynch, The Byzantine Firing Squad Problem, *MIT Technical Report, MIT/LCS/TM–275*, April 1985.

[CM] K. M. Chandy and J. Misra, How processes learn, *Distributed Computing*, **1**:1, 1986, pp. 40-52.

[C] B. Coan, A Communication-Efficient Canonical Form for Fault-Tolerant Distributed Protocols, *Proceedings of the Fifth PODC*, 1986, pp. 63-72.

[CDDS] B. Coan, D. Dolev, C. Dwork, and L. Stockmeyer, The distributed firing squad problem, *Proceedings of the Seventeenth STOC,* 1985, pp. 335-345.

[DRS] D. Dolev, R. Reischuk, and H. R. Strong, Eventual is earlier than immediate, *Proceedings of the 23th FOCS,* 1982, pp. 196-203.

[DM] C. Dwork and Y. Moses, Knowledge and common knowledge in a Byzantine environment: The case of crash failures. *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, Monterey, 1986, J.Y. Halpern ed., Morgan Kaufmann, pp. 149-170. Slightly revised as *MIT Technical Report, MIT/LCS/TM-300*, July 1986.

[F] M. J. Fischer, The consensus problem in unreliable distributed systems (a brief survey), *Yale University Technical Report YALEU/DCS/RR-273*, 1983.

[FI] M. J. Fischer and N. Immerman, Foundations of knowledge for distributed systems, *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, Monterey, 1986, J.Y. Halpern ed., Morgan Kaufmann, pp. 171-185.

[FL] M. J. Fischer and N. A. Lynch, A lower bound for the time to assure interactive consistency, *Information Processing Letters*, **14**:4, 1982, pp. 183-186.

[GJ] M. R. Garey and D. S. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*, W. H. Freeman and Company, San Francisco, 1979.

[Ha] V. Hadzilacos, A lower bound for Byzantine agreement with fail-stop processors, *Harvard University Technical Report TR-21-83.*

[HF] J. Y. Halpern and R. Fagin, A formal model of knowledge, action, and communication in distributed systems, *Proceedings of the Fourth PODC,* 1985, pp. 224-236.

[HM] J. Y. Halpern and Y. Moses, Knowledge and common knowledge in a distributed environment, Version of January 1986 is available as IBM research report *RJ 4421.* Early versions appeared in *Proceedings of the Third PODC*, 1984, pp. 50–61; and as IBM research report *RJ 4421*, 1984.

[HM2] J. Y. Halpern and Y. Moses, A guide to the modal logic of knowledge and belief, *Proceedings of the Ninth IJCAI,* 1985, pp. 480-490.

[Hi] J. Hintikka, *Knowledge and Belief*, Cornell University Press, 1962.

[HU] J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, Massachusetts, 1979.

[LF] L. Lamport and M. J. Fischer, Byzantine grenerals and transaction commit protocols, *SRI Technical Report Op.62*, 1982.

[Me] M. Merritt, Notes on the Dolev-Strong lower bound for Byzantine agreement, unpublished manuscript, 1985.

[Mi] R. Michel, Attaining common knowledge in synchronous distributed networks, unpublished manuscript, 1986.

[Mo] Y. Moses, Knowledge in a distributed environment, *Ph.D. Thesis, Stanford University Technical report STAN-CS-1120*, 1986.

[MSF] C. Mohan, H. R. Strong, and S. Finkelstein, Methods for distributed transaction commit and recovery using Byzantine agreement within clusters of processors, *Proceedings of the Second PODC,* 1983, pp. 89–103.

[PR] R. Parikh and R. Ramanujam, Distributed processes and the logic of knowledge (preliminary report), *Proceedings of the Workshop on Logics of Programs*, 1985, pp. 256-268.

[PSL] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, *JACM*, **27**:2, 1980, pp. 228-234.

[R] M. O. Rabin, Efficient solutions to the distributed firing squad problem, private communication.