Programming Simultaneous Actions Using Common Knowledge: Preliminary Version*

Yoram Moses and Mark R. Tuttle

MIT Laboratory for Computer Science
Cambridge, Massachusetts 02139

Abstract: This work applies the theory of knowledge in distributed systems to the design of fault-tolerant protocols for problems involving coordinated simultaneous actions in synchronous systems. We give a simple method for transforming specifications of such problems into high-level protocols programmed using explicit tests of whether certain facts are common knowledge. The resulting protocols are *optimal in all runs*: for every possible input to system and pattern of processor failures, they are guaranteed to perform the simultaneous actions as soon as any other protocol can possibly perform them. A careful analysis of when facts become common knowledge shows how to efficiently implement these protocols in many variants of the omissions failure model. In the generalized omissions model, however, it is shown that any protocol that is optimal in this sense must require co-NP hard computations. The analysis in this paper exposes subtle differences between the failure models, including the precise point at which this gap in complexity occurs.

1 Introduction

The problem of ensuring proper coordination between processors in distributed systems with unreliable components 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 the system. It is therefore of great interest to study the design of protocols involving simultaneous actions.

This paper presents a novel approach to the design of fault-tolerant protocols for performing coordinated simultaneous actions in synchronous systems, for a number of variants of the *omissions* failure model (cf. [MSF]). We define a general notion of a *simultaneous choice problem*, which is intended to capture the essence of simultaneous coordination in such a system. Many well-known problems, such as simultaneous Byzantine agreement, distributed firing squad, etc. can be formulated as such problems. Given a satisfiable specification of a simultaneous choice problem, we derive a protocol for the problem with the unique property of being *optimal in all runs*: For every possible input to the system and pattern of faulty processor behavior, this protocol is guaranteed to perform the simultaneous actions as soon as they would be performed under any other correct protocol for the problem. In contrast, most previous protocols for coordinated actions in unreliable systems do not adapt their behavior based on the pattern of failures, and hence always perform as poorly as they do in their *worst case* run. (We will often use *optimal* as shorthand for optimal in all runs.)

Our approach is based on the close relationship between knowledge, communication and action in distributed systems. More specifically, a number of recent works (cf. [HM],[DM],[Mo]) point out 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. Roughly speaking, every processor performing a simultaneous action — an action that is guaranteed to be performed

*The first author's affiliation as of 1987 will be the department of Applied Mathematics, Weizmann Institute, Rehovot, 76100, Israel. His work 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.

0272-5428/86/0000/0208$01.00 © 1986 IEEE

by all of the processors simultaneously whenever it is performed at all — *knows* that the action is being performed. It follows that every processor knows that all other processors know that the action is being performed. This argument can be formalized and completed to show that when a simultaneous action is performed, all relevant processors must have common knowledge that it 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 correspondence between common knowledge and simultaneous actions. The derivation is done in two stages. In the first stage we program the protocol in a high-level language in which processors' actions depend on explicit tests of whether certain facts are common knowledge (cf. [DM],[HF]). These high-level protocols are automatically extracted from the problem specification via a few simple manipulations. For example, consider the following simple version of the *distributed firing squad* problem (cf. [BL],[CDDS],[R]): An external source may send "start" messages to some of the processors in the system, at unpredictable, possibly different, times. It is required that (i) if some nonfaulty processor receives a "start" message, all nonfaulty processors should simultaneously perform an irreversible "firing" action at some later point, (ii) whenever any processor "fires" all of the nonfaulty processors do, and (iii) if no processor receives a "start" message, then no "firing" occurs. The high-level protocol we derive for this problem requires all processors to act as follows:

```
In every round do:

  if  it is common knowledge that some
        processor received a "start" message
     then
        "fire" and halt
     else
        send current view to every processor.
```

The second stage in the derivation consists of designing effective methods for implementing tests for common knowledge. We present a uniform method for effectively implementing these tests, in all of the variants of the omissions failure model we consider. This provides a way of compiling such high-level protocols into low-level standard protocols in these failure models. As a consequence, for example, the above protocol yields an optimal protocol for the distributed firing squad problem. 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. A general method for obtaining optimal protocols for simultaneous problems in the simpler crash failure model is implicit in the work of Dwork and Moses (cf. [DM]), which provided the original motivation for this work.

We show that optimal protocols for simultaneous choice problems in all variants of the omissions model can always be implemented in a communication efficient way. However, it turns out that a naive method of implementing 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 in investigating methods of efficiently implementing tests for common knowledge in the different variants of the omissions 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 exposes some of the basic structure of 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, the problem is shown to be trivial. This exposes a big difference between two seemingly symmetric models.

We are not able to efficiently implement tests for common knowledge in the *generalized omissions* model, in which an undelivered message implies only that either the sender or the intended receiver is faulty. This, it turns out, is not a coincidence. In fact, we show that unless P=NP, no optimal protocol for any non-trivial simultaneous problem in this model can be computationally efficient. We prove that any such protocol must require the participating processors to perform co-NP hard computations between consecutive rounds of communication. In particular, there can be no computationally efficient optimal protocol for the distributed firing squad problem stated above, for simultaneously performing Byzantine agreement (cf. [PSL],[DM]), and for most any other simultaneous problem in this model. 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, yielding computationally efficient optimal protocols. We are therefore able to precisely identify the point at which optimal protocols for simultaneous actions become computationally prohibitive.

The remainder of the paper is organized as follows: Section 2 defines the models of distributed systems used in the paper, and Section 3 gives a precise definition 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 requiring coordinated simultaneous action. Section 5 presents a uniform method for 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 of whether facts relevant to simultaneous choice problems are common knowledge. The analysis in Section 6 reveals interesting properties of the different failure models, and exposes fine distinctions between them. Finally, Section 7 contains concluding remarks and suggestions for future work.

2 Model of a System

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

We consider a synchronous distributed system consisting of a finite collection $P = \{p_1, \ldots, p_n\}$ of $n \geq 2$ processors (automata), each pair of which is connected by a two-way communication link. The processors share a discrete global clock that starts out at time 0 and advances by increments of one.[1] Communication in the system proceeds in a sequence of *rounds*, with round k taking place between time $k-1$ and time k. In each round, every processor first sends messages it needs to send to other processors, and then receives messages sent to it by other processors in the same round. The identity of the sender and destination of each message, as well as the round in which it is sent, are assumed to be part of the message. The processors in the system communicate among themselves, as well as with elements external to the system (*clients*), who may make various requests of the system (think, for example, of a distributed airline reservation system). A processor p starts out in some *initial state* σ. At any given time, a processor's *input history* consists of its initial state together with the requests it received from the system's (external) clients.

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

Similarly, a processor's *message history* consists of the set of messages it has received from other processors. A processor's *view* at any given time consists of its input history, message history, and the time on the global clock.

We think of the processors as following a *protocol*, which specifies exactly what messages each processor is required to send (and what other actions the processor should take) at each round, as a *deterministic* function of the processor's view. However, processors are unreliable, and thus some of them might be *faulty*, the rest being *nonfaulty*. Both faulty and nonfaulty processors faithfully follow their protocol, but their behaviors differ in the messages they succeed in sending and receiving. A nonfaulty processor succeeds in sending *all* of the messages it is required by the protocol to send, and receives all of the messages sent to it. We will distinguish a number of different models of faulty processor behavior: (i) The *omissions* model (cf. [MSF]), in which a faulty processor receives all of the messages sent to it but succeeds in sending 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 succeeds in receiving only an arbitrary subset of the messages sent to it, but succeeds in sending *all* of the messages it is required to send; (iii) the *generalized omissions* model, in which a faulty processor both succeeds in sending only an arbitrary subset of the messages it is required to send and in receiving 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 that does not receive a message from another processor can determine whether it or the sender is at fault. (Formally, this is modeled by the processor receiving an "error" message to that effect.)

An infinite execution of a protocol is called a *run* of the protocol. A run of a given protocol can be uniquely specified by presenting a complete history of the events that take place, from time 0 until the end of time. This includes each processor's complete input history, message history, and, if the processor is faulty in the particular run, its precise behavior in each round. Formally, a *faulty behavior sequence* for a processor in the omissions model is simply a sequence $\langle S_1, S_2, \ldots \rangle$ of sets of processors. A processor is said to *display* such a faulty behavior sequence in a given run if in every round k of the run the processor fails to send the messages it is required to send to processors in S_k, and succeeds in sending all such messages to processors not in S_k. A *failure pattern* of a run is a set of pairs $(p_i, \langle S_1^i, S_2^i, \ldots \rangle)$ consisting of a processor and a faulty behavior sequence, such that the processors appearing in the failure pattern are exactly those that are faulty in the run, and they each display the corresponding faulty behavior sequence. The notion of a failure pattern can be similarly defined for the other models of failure we have mentioned. In the receiving omissions model the sets S_k are replaced by sets of processors R_k from which the processor fails to receive messages, and in the variants of generalized omissions they are replaced by pairs $\langle S_k, R_k\rangle$ of sets of processors. Given $\gamma_1, \ldots, \gamma_n$, where γ_i is processor p_i's complete input history in a given run ρ, the *(external) input* to ρ is simply $\gamma = (\gamma_1, \ldots, \gamma_n)$. A pair (π, γ), where π is a failure pattern and γ is an input, is called an *operating environment*. A run is uniquely determined by a protocol and an 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 — runs with the same operating environments.

Our purpose is to study the behavior of protocols in the presence of a bounded number of failures of a particular type, in a given setting of possible inputs. Thus, we identify a system with the set of all possible runs of a given protocol under such circumstances. Formally, a system is determined by a protocol $\mathcal{P}$, a failure model, natural numbers $n \geq 2$ and $t \leq n-2$, and sets Γ_i of individual processor inputs, $i \leq n$. The system is identified with the set of all runs of $\mathcal{P}$ by n processors, at most t of which are faulty (in the sense of the given failure model), and in which every processor p_i's input (initial state and external messages) is some $\gamma_i \in \Gamma_i$. Thus, the set of possible inputs in the system has the form $\Gamma_1 \times \Gamma_2 \times \cdots \times \Gamma_n$. This definition ensures that the external input to the system is orthogonal to, and hence carries no information about, the failure pattern. Furthermore, it ensures that one processor's external input contains no information about other processors' external input.

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. Roughly speaking, a point corresponds to an instantaneous description of the system. We denote processor p's view at the point (ρ, ℓ) is denoted $v(p, \rho, \ell)$.

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 presents precise definitions of the types of knowledge we will deal with. Our treatment is a modification of that of [DM] and [HM].

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. We assume the existence of an underlying logical language for representing all of the relevant *ground* facts — facts about the system that do not explicitly mention processors' knowledge; e.g., *"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 $\tau(\varphi)$. 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 the 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 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, 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 that two points (ρ, k) and (ρ', k') are *p_i-equivalent*, denoted $(\rho, k) \overset{i}{\leftrightarrow} (\rho', k')$, iff $v(p_i, \rho, k) = v(p_i, \rho', k')$. (Given that the global time is defined to be part of a processors view, it turns out that $(\rho, k) \overset{i}{\leftrightarrow} (\rho', k')$ implies that $k = k'$.) We say that a processor p_i *knows* a fact φ at (ρ, k), denoted $(\rho, k) \models K_i\varphi$, if $(\rho', k) \models \varphi$ for all points (ρ', k) satisfying $(\rho, k) \overset{i}{\leftrightarrow} (\rho', k)$. 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 its computational power or on internal actions it performs.

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:

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

Thus, roughly speaking, G's view is simply the joint view of its members. We say that *the group G has implicit knowledge* of φ at (ρ, k), denoted $(\rho, k) \models I_G\varphi$, if for all runs ρ' 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 a singleton set $\{p_i\}$, the notions of I_G and K_i coincide. Intuitively, G has implicit knowledge of 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 have implicit knowledge of ψ, even if neither of them knows ψ individually. The notion of implicit knowledge was first defined in [HM].

Finally, the state of *common knowledge* among a group of processors will be central to our analysis. The kind of groups of interest will not always be 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 (i.e., run), this group is a fixed set G of processors. But the precise identity of $\mathcal{N}$ varies from one context to another. This motivates us to define common knowledge for a slightly more general notion of a group 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. Given an indexical set $\mathcal{S}$, we define $E_{\mathcal{S}}\varphi$, essentially corresponding to *everyone in* $\mathcal{S}$ *knows* φ, by:

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

Roughly speaking, $E_{\mathcal{S}}\varphi$ holds exactly if every member of $\mathcal{S}$ knows that if it is a member of $\mathcal{S}$ then φ holds. Notice that if $\mathcal{S}$ is a fixed set G, then $p_i \in \mathcal{S}$ iff $K_i(p_i \in \mathcal{S})$, and hence $E_{\mathcal{S}}\varphi \equiv E_G\varphi$ is valid in the system.

We can 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 $(\rho, \ell) \models E_{\mathcal{S}}^m\varphi$. The definitions of $E_{\mathcal{S}}$ and $C_{\mathcal{S}}$ directly generalize the standard notions from [HM] and [DM].

We now mention some of the properties of the notions of knowledge defined above. An operator M is said to satisfy 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, implicit knowledge, and common knowledge given above immediately imply the following proposition (cf. [HM2], [DM]):

Proposition 1: The operators K_i, I_G and $C_{\mathcal{S}}$ each satisfy the modal system S5.

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 the points are p_i-equivalent for some processor $p_i \in \mathcal{S}(\rho, k) \cap \mathcal{S}(\rho', k)$. 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 who is nonfaulty at *both* points, which has the same view at both points. It can now be shown that $(\rho, k) \models E_{\mathcal{S}}^m\varphi$ iff $(\rho', k) \models \varphi$ for all points (ρ', k) of distance $\leq m$ from (ρ, k) in this graph. 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 this graph. The indexical set $\mathcal{S}$ is generally clear from context (usually being the set $\mathcal{N}$ of nonfaulty processors). We thus denote similarity by $\sim$ without the superscript $\mathcal{S}$. We can now show:

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

One of the useful properties of common knowledge is:

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

which implies that when a fact becomes common knowledge all relevant processors simultaneously come to know that it is common knowledge. Another useful fact about common knowledge is captured by the following rule, which roughly states that facts that are "public" are common knowledge:

if $\varphi \supset E_{\mathcal{S}}\varphi$ is valid in the system
then $\varphi \supset C_{\mathcal{S}}\varphi$ is valid in the system.

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., about runs. A fact φ is said to be a fact *about the run* if fixing the run determines whether or not φ is true. That is, if for all runs ρ and times k and ℓ it is the case that $(\rho, k) \models \varphi$ iff $(\rho, \ell) \models \varphi$. The meaning of a fact being *about the input*, *about the operating environment*, *about the past*, *about the first k rounds*, etc., are similarly defined.

4 Simultaneous Choice

In order to study the design of protocols for problems involving coordinated simultaneous actions, we need a definition of this class of problems. Lacking a most general definition of such problems, we focus on the class of *simultaneous choice* problems, a large class of problems that capture the essence of such coordinated actions in a distributed environment. Roughly speaking, these problems require that all of the relevant processors identically choose one of a number of possible alternative actions, where for every action we are given conditions under which the action *must* be performed and conditions under which its performance is forbidden. A specification of such a problem must, in addition, determine the possible settings (i.e., possible external inputs) in which the protocol may be required to perform the choice.

Formally, a *simultaneous action* is an action a together with associated conditions $pro(a)$ and $con(a)$, which are conditions on the operating environment. A *simultaneous choice problem* $\mathcal{C}$ is specified by a set of actions $\{a_1, \dots, a_m\}$, with their associated conditions, together with sets Γ_j of possible inputs to processor p_j. A protocol $\mathcal{P} = \mathcal{P}(n,t)$ *implements* $\mathcal{C}$ if every run ρ of the systems defined by $(\mathcal{P}(n,t), n, t, \{\Gamma_j\}_{j \leq n})$ satisfies the following conditions: (i) at most one of the a_i's is performed, (ii) any a_i performed is performed simultaneously by all nonfaulty processors, (iii) a_i is performed if ρ satisfies $pro(a_i)$, and (iv) a_i is not performed if ρ satisfies $con(a_i)$. We say that $\mathcal{C}$ is *implementable* if such a protocol $\mathcal{P}$ exists. Denoting by $\Psi(a_i)$ the property of being a run in which a_i is performed, it is easy to see that if $\mathcal{P}$ implements $\mathcal{C}$ then the following formulas are valid in the system: $pro(a_i) \supset \Psi(a_i)$, $con(a_i) \supset \neg\Psi(a_i)$, and $\Psi(a_i) \supset \neg\Psi(a_j)$ $(j \neq i)$. We also consider a closely related class of problems, called *strict* simultaneous choice problems, which are defined as simultaneous choice problems are, except that runs of an implementing protocol are required to satisfy the modified condition (i′) *exactly* one of the a_i's is performed, together with the conditions (ii)-(iv) above. We note that 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 similarly stated in terms of any (indexical) set of processors, even the set P of all processors, with the analysis in this section and the next one carrying through without change.

Many familiar problems requiring simultaneous action by a group of processors are instances of a simultaneous choice or strict simultaneous choice. In all known instances, the conditions $pro(a_i)$ and $con(a_i)$ 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 protocols allow the nonfaulty processors to display default behavior in the presence of failures; cf. [LF].) For example, the distributed firing squad problem is a simultaneous choice consisting of the "firing" action a, the condition $pro(a)$ being the receipt of a "start" signal by a nonfaulty processor, and the condition $con(a)$ being that no processor receives a "start" signal. The associated sets Γ_i of inputs simply allow for a "start" message to be delivered to any processor at any time. The simultaneous Byzantine agreement problem (cf. [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". The condition $pro(a_0)$ is that all initial values are 0, and the condition $pro(a_1)$ is that all initial values are 1. (The conditions $con(a_0)$ and $con(a_1)$ are both taken to be *false*.) The associated sets Γ_i each 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. Notice that for most assignments of initial values, deciding either 0 or 1 is acceptable according to the problem specification. Simultaneous Byzantine agreement is a *strict* simultaneous choice, since the processors are required to decide either 0 or 1 in every run.

We are now in a position to formally state the relationship between simultaneous action and common knowledge mentioned in the introduction: When a simultaneous choice is performed, it is common knowledge that the choice is being performed.

Lemma 3: Let ρ be a run of a protocol $\mathcal{P}$ implementing a simultaneous choice (resp. strict simultaneous choice) among $\{a_1, \dots, a_m\}$. If a_i is performed at time ℓ in ρ, then $(\rho, \ell) \models C_{\mathcal{N}} \Psi(a_i)$.

5 Optimal Protocols

In this section, we show how a high-level, optimal protocol for a simultaneous choice problem can be extracted directly from the problem specification. We start by considering a very simple protocol $\mathcal{F}$ that will serve as a basic building block in such optimal protocols:

for $\ell \geq 0$, at time ℓ each processor sends its current view to every other processor.

This protocol is called the *full-information protocol* (cf. [H], [FL], [PSL]). Intuitively, since $\mathcal{F}$ requires processors to send all of the information available to them at each point during a run, one would expect this protocol to give each processor as much information about the operating environment as any protocol would. In fact, the next lemma shows that if a processor can not distinguish two operating environments during runs of $\mathcal{F}$, then the processor can not distinguish these operating environments during runs of any other protocol.

Lemma 4: Let ρ, ρ' be runs of $\mathcal{F}$ and ς, ς' be runs of a protocol $\mathcal{P}$ such that ρ and ς (resp., ρ' and ς') have the same operating environment. If $(\rho,\ell) \overset{i}{\leftrightarrow} (\rho',\ell)$ then $(\varsigma,\ell) \overset{i}{\leftrightarrow} (\varsigma',\ell)$.

The following corollary of Lemma 4 captures the generality of the protocol $\mathcal{F}$ in a precise sense. The first part says that $\mathcal{F}$ is as good as any protocol for perpetuating knowledge about the operating environment. More importantly, the second part says that facts about the operating environment become common knowledge to the nonfaulty processors during runs of $\mathcal{F}$ at least as early as they do during runs of any other protocol.

Corollary 5: Let φ be a fact about the operating environment, and let ρ be a run of $\mathcal{F}$ and ς be a run of a protocol $\mathcal{P}$ such that ρ and ς have the same operating environment. Then

a) If $(\varsigma,\ell) \models K_i\varphi$ then $(\rho,\ell) \models K_i\varphi$.

b) If $(\varsigma,\ell) \models C_{\mathcal{N}}\varphi$ then $(\rho,\ell) \models C_{\mathcal{N}}\varphi$.

We now show how $\mathcal{F}$ can be used as the basis of an optimal protocol for any implementable simultaneous choice. From the definition of a simultaneous choice we see that performing the action a_i during a run is forbidden iff the run satisfies either $con(a_i)$ or one of the $pro(a_j)$'s (for $j \neq i$). Let us define

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

A run satisfies $enabled(a_i)$ iff performing a_i during the run is not forbidden by the specification of the problem. If $enabled(a_i)$ is common knowledge at a point (ρ,ℓ), then it is common knowledge at (ρ,ℓ) that performing a_i is not forbidden. Conversely, if the action a_i is performed at the point (ρ,ℓ), then Lemma 3 implies that $\Psi(a_i)$ is common knowledge at (ρ,ℓ). Since $\Psi(a_i) \supset enabled(a_i)$, it follows that $enabled(a_i)$ is also common knowledge at (ρ,ℓ). This motivates consideration of the following protocol $\mathcal{P}_C$:

```
for $\ell \geq 0$, at time $\ell$ perform the following:
  if $C_{\mathcal{N}} enabled(a_i)$ holds for some $a_i$
    then
      let $i = \min\{j : C_{\mathcal{N}} enabled(a_j) \text{ holds}\}$;
      perform $a_i$;
      halt
    else
      send current view to every processor.
```

The protocol $\mathcal{P}_C$ is not a protocol in the usual sense since a processor's actions depend on whether certain facts are common knowledge (cf. [DM]; see [HF] for a similar notion of *knowledge-based protocols*). Protocols in which processors' actions do not depend on explicit tests for knowledge or common knowledge of certain facts are called *standard* protocols (termed *simple* protocols in [HF]). Notice that $\mathcal{P}_C$ halts when $\bigvee_i C_{\mathcal{N}} enabled(a_i)$ first holds. Corollary 5 implies that this holds in runs of $\mathcal{F}$ as early as it holds in runs of any other protocol. Intuitively, since $\mathcal{P}_C$ sends messages precisely as required by $\mathcal{F}$ until an action is performed, the same should be true in runs of $\mathcal{P}_C$. We formalize this in the following lemma.

Lemma 6: Let ρ be a run of $\mathcal{F}$ and let ς be a run of $\mathcal{P}_C$ such that ρ and ς have the same operating environment. If $(\varsigma,\ell) \not\models \bigvee_i C_{\mathcal{N}} enabled(a_i)$ then $(\rho,\ell) \not\models \bigvee_i C_{\mathcal{N}} enabled(a_i)$.

Finally, using Lemma 3, Corollary 5, and Lemma 6, we can show that $\mathcal{P}_C$ is an optimal protocol:

Theorem 7: If $\mathcal{C}$ is an implementable simultaneous choice (resp., strict simultaneous choice), then $\mathcal{P}_C$ is an optimal protocol for $\mathcal{C}$.

Thus, by Lemma 6 and Theorem 7, the problem of optimally performing simultaneous choice problems can be reduced to the problem of determining when facts such as $enabled(a_i)$ become common knowledge during runs of $\mathcal{F}$. In the remainder of the paper we restrict our attention exclusively to the protocol $\mathcal{F}$. Recall that a fundamental property of $\mathcal{F}$ is that processors are required to send their entire view in every round. Since, strictly speaking, a processor's view is exponential in size, $\mathcal{F}$ seems to require processors to send messages of exponential length. We now show that there is a simple, compact representation of a processor's view that may be sent instead.

Given a run ρ, the *communication graph* (cf. [Me]; see Figure 1) of ρ represents what messages are delivered in ρ. It is a layered graph (with one layer corresponding to every natural number) in which every processor in the system is represented by one node in every layer. A portion of a communication graph is shown in Figure 1. We denote the node representing p_i at time ℓ by $\langle p_i,\ell\rangle$. Edges connect nodes in adjacent layers, with an edge between $\langle p_i, k-1\rangle$ and $\langle p_j, k\rangle$ iff a message from p_i is delivered to p_j in round k. The *labeled communication graph* is obtained by labeling a layer 0 node of the communication

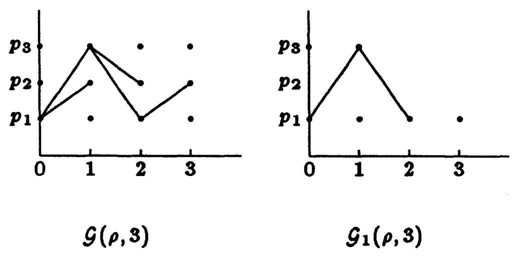

Figure 1: Communication graphs.

graph by the processor's initial state, and the other nodes by the inputs the processor receives from external clients. We note in passing that the labeled communication graph of a run of $\mathcal{F}$ is a representation of the operating environment of the run. 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 the run ρ.

Informally, at every point (ρ, ℓ), a processor p_i's view corresponds to a certain subgraph $\mathcal{G}_i(\rho, \ell)$ of $\mathcal{G}(\rho, \ell)$. For example, $\mathcal{G}_1(\rho, 3)$ is shown in Figure 1 for a particular run ρ. The graphs $\mathcal{G}_i(\rho, \ell)$ are easily definable in terms of $\mathcal{G}(\rho, \ell)$. Details are left to the full paper. The next lemma states that the labeled communication graph corresponding to a processor's view at a point uniquely determines its view at the point.

Lemma 8 : For all runs ρ and ρ' of $\mathcal{F}$, $v(p_i, \rho, \ell) = v(p_i, \rho', \ell)$ iff $\mathcal{G}_i(\rho, \ell) = \mathcal{G}_i(\rho', \ell)$.

Consequently, a processor's view of the run and the processor's view of the corresponding labeled communication graph convey the same information. 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 messages sent to the system by its clients. Thus, if we require processors to send their view of the labeled communication graph instead of their view of the run, then messages required by $\mathcal{F}$ are only polynomial in size.[2] In addition, a processor receiving such messages in a given round can easily construct the labeled communication graph corresponding to its view at the end of the round in only polynomial time. Thus, the use of such compact representations of a processor's view is also computationally efficient.

[2]In the Byzantine failure models in which processors are allowed to lie, however, such compact representations are not guaranteed to exist; cf. [C].

6 Testing Common Knowledge

Programming protocols using tests for common knowledge is a very powerful programming technique: We have reduced the problem of designing an optimal protocol for a simultaneous choice to the problem of testing for common knowledge of certain facts. However, since the resulting protocols involve tests for common knowledge, it is not immediately obvious that they can be *compiled* into standard protocols that can be followed by conventional processors. We now show that this is generally possible. However, in order to do so we need to slightly restrict the class of "acceptable" simultaneous choices by requiring that the conditions *enabled*(a_i) be decidable. Using "$\mathcal{G}(\rho, \ell)$" to denote the property of being a run having $\mathcal{G}(\rho, \ell)$ as a prefix of its labeled communication graph, a fact φ is said to be *effective* if there is a deterministic algorithm for determining whether "$\mathcal{G}(\rho, \ell) \supset \varphi$" is valid in the system, given $\mathcal{G}(\rho, \ell)$ as input. For all natural simultaneous choice problems it is the case that the conditions *enabled*(a_i) are effective facts. We say that a simultaneous choice is *effective* if each condition *enabled*(a_i) is effective.

Suppose, now, that φ is an effective fact. Then it is possible to effectively determine whether φ is common knowledge at a point (ρ, ℓ) by enumerating all (exponentially many) points (ρ', ℓ), testing whether each of them is similar to (ρ, ℓ), and determining whether or not φ holds at each of these points. Thus, we have:

Theorem 9 : For all variants of the omissions model, if C is an effective, implementable simultaneous choice, then P_C may be compiled into a standard optimal protocol for C.

Clearly, implementing tests for common knowledge of certain facts by the method described above is very inefficient. The rest of the paper is devoted to investigating ways of implementing such tests in a computationally efficient manner in the different failure models. In order to do so, we must further restrict the class of "acceptable" simultaneous choices to require that computing whether *enabled*(a_i) is determined by a subgraph of the labeled communication graph is computationally tractable. We say that a fact φ is *practical* if it is a fact about the input and existence of failures for which there is an algorithm determining the validity of "$\mathcal{G}_i(\rho, \ell) \supset$ *enabled*(a_i)" in polynomial time. A simultaneous choice is *practical* if each condition *enabled*(a_i) is practical. Notice that all natural simultaneous choice problems are practical. We note that if C is a practical simultaneous choice, then the above mentioned method can be used to compile P_C

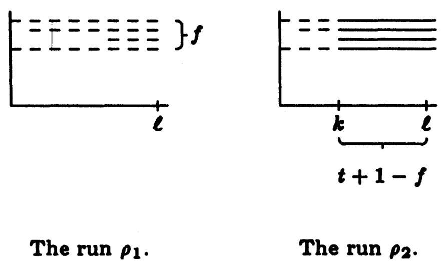

Figure 2: Runs illustrating Lemma 10.

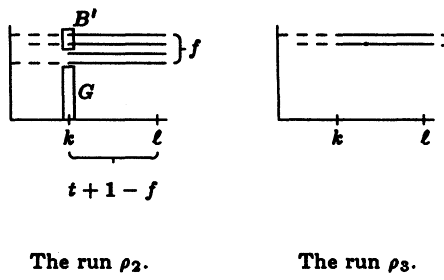

Figure 3: Runs illustrating Lemma 11.

into a PSPACE implementation of $\mathcal{C}$. However, as we will see in the rest of this section, it is often possible to do much better.

6.1 The Omissions Model

In this subsection, we develop an efficient construction that precisely characterizes what facts are common knowledge in a run of $\mathcal{F}$ in the omissions failure model. This construction gives rise to computationally-efficient, optimal protocols implementing simultaneous choices. The construction is arrived at as a result of a careful analysis of what facts are *not* common knowledge at a point (ρ, ℓ). This construction is motivated by the next two lemmas.

We say that a processor is *silent from time k* if it sends no messages from time k. A processor is said to *fail* before time ℓ if it displays faulty behavior before time ℓ. Now, 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 by time ℓ. In the following lemma we show that $(\rho_1, \ell) \sim (\rho_2, \ell)$ where ρ_2 differs from ρ_1 only in that the faulty processors are silent in ρ_2 from time k, where $k = \ell - (t + 1 - f)$.

Lemma 10: Let ρ_1 be a run in which at most f processors fail before time ℓ. Let ρ_2 be a run differing from ρ_1 only in that processors failing before time ℓ 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 proof of Lemma 10 follows from a technical lemma similar to Lemma 15 of [DM]. Details are left to the full paper. Lemma 10 implies, for instance, that the views at time k of processors failing before time ℓ in ρ_1 are not common knowledge at time ℓ since these processors are silent from time k in ρ_2. In addition, no fact about the input not determined by time k is common knowledge.

Given a point (ρ, k) and a set of processors G, we define

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

The essence of the second lemma is captured by the runs ρ_2 and ρ_3 of Figure 3. In ρ_2, the faulty processors are silent from time k. G is the set of nonfaulty processors and $B' = B(G, \rho, k)$. The run ρ_3 differs from ρ_2 in that the processors in $P - B'$ do not fail in ρ_3. The following lemma states that $(\rho_2, \ell) \sim (\rho_3, \ell)$ and, in addition, that the processors in G have the same views at time k in both ρ_2 and ρ_3. Formally, we have (see Figure 3):

Lemma 11: Let ρ_2 be a run in which the f processors that fail 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 a run that differs from ρ_2 in that processors in $P - B'$ do not fail. Then

a) $(\rho_2, \ell) \sim (\rho_3, \ell)$, and

b) $v(G, \rho_2, k) = v(G, \rho_3, k)$.

Thus, for instance, the failure of the processors in $P - B'$ can not be common knowledge at (ρ_2, ℓ) since they do not fail in ρ_3.

Going back to Figures 2 and 3, notice that if $f' < f$ then, setting $\rho_1' = \rho_3$, Lemmas 10 and 11 can be applied again. Iterating this process (at most t times) we reach a run $\hat{\rho}$ satisfying $(\rho_1, \ell) \sim (\hat{\rho}, \ell)$ in which the $\hat{f}$ faulty processors are silent from time $\hat{k} = \ell - (t+1-\hat{f})$, and at $(\hat{\rho}, \hat{k})$ all the faulty processors are implicitly known to be faulty by the nonfaulty processors. We are about to show that the view of the nonfaulty processors at $(\hat{\rho}, \hat{k})$ will characterize what facts are common knowledge at (ρ_1, ℓ). This is shown by considering what happens when individual processors perform an analogous iterative construction based on their individual views. We now formalize such a local construction, illustrated in Figure 4.

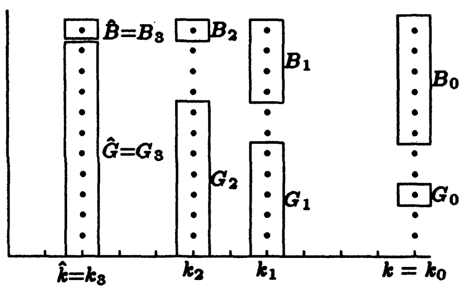

Figure 4: An example of the construction for $t = 9$.

Let $G_0 = \{p_j\}$ and $k_0 = \ell$. For $i \geq 0$, we define $B_i = B(G_i, \rho, k_i)$, and we define G_{i+1} and k_{i+1} inductively as follows:

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

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

If k_i ever becomes negative, then for all $j \geq i$ we define $k_j = k_i$ and we define G_j and B_j to be empty.

This construction must halt within t iterations, and we can show that the results of the construction (the limits of the sequences $\{G_i\}$, $\{B_i\}$, and $\{k_i\}$) are independent of the processor p_j with which the construction began. We denote the results of the construction by $\hat{G}$, $\hat{B}$, and $\hat{k}$. We define $\hat{V}(\rho,\ell)$ to be $v(\hat{G}, \rho, \hat{k})$ if $\hat{k} \geq 0$, and empty otherwise. Since each processor is able to compute $\hat{V}(\rho,\ell)$ independently, $\hat{V}(\rho,\ell)$ is common knowledge at (ρ,ℓ). Conversely, Lemmas 10 and 11 can be used to show that $\hat{V}(\rho,\ell)$ completely characterizes the connected component of (ρ,ℓ) in the similarity graph. That is:

Lemma 12: $\hat{V}(\rho,\ell) = \hat{V}(\rho',\ell)$ iff $(\rho,\ell) \sim (\rho',\ell)$.

Consequently, the set $\hat{V}(\rho,\ell)$ completely characterizes the set of facts that are common knowledge at the point (ρ,ℓ). More precisely we have the following:

Theorem 13: $(\rho,\ell) \models C_{\mathcal{N}}\varphi$ iff $(\rho',\ell) \models \varphi$ for all ρ' satisfying $\hat{V}(\rho,\ell) = \hat{V}(\rho',\ell)$.

As a result, facts about the input and existence of failures that are common knowledge at the point (ρ,ℓ) must follow directly from the set $\hat{V}(\rho,\ell)$, as we see in the following corollary. Using "V" to denote the property of being a run having V as a set of views, we have the following:

Corollary 14: If φ is a fact about the input and the existence of failures, then $(\rho,\ell) \models C_{\mathcal{N}}\varphi$ iff "$V \supset \varphi$" is valid in the system for $V = \hat{V}(\rho,\ell)$.

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. We note that a processor knows that another processor is faulty iff it knows of a message the processor failed to send and this is an easy fact to check given the communication graph corresponding to the processor's view. Since the construction of $\hat{V}(\rho,\ell)$ halts after t iterations, the construction can be computed locally in polynomial time. By Corollary 14 we now have the following:

Theorem 15: If $\mathcal{C}$ is a practical, implementable simultaneous choice, then $\mathcal{P}_{\mathcal{C}}$ can be compiled into a polynomial-time, standard protocol for $\mathcal{C}$.

We reiterate the fact that $\mathcal{P}_{\mathcal{C}}$ is a protocol for $\mathcal{C}$ that is optimal in *all runs*: actions are performed in runs of $\mathcal{P}_{\mathcal{C}}$ as soon as they can possibly be performed by any protocol in runs having the same operating environment. Thus, for example, simultaneous Byzantine agreement is performed in anywhere between two 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 one 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 early in the run. In particular, if there are no failures, no fact about the input is common knowledge less that $t+1$ rounds after it is first determined to hold.

The fact that the information in $\hat{V}(\rho,\ell)$ is essentially all that is common knowledge at a given point has interesting implications. For example, recall that in the traditional simultaneous Byzantine agreement or consensus problems (cf. [PSL], [F], [DM]), the processors are only required to decide, say, v in case they all started with an initial value of v. It would be more pleasing, however, if they would decide v whenever the majority of initial values were v. This is clearly impossible, since some processors may be silent throughout the run. However, by choosing the majority of the initial values appearing in $\hat{V}(\rho,\ell)$ as their decision value, the processors can 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 can be bad only in runs in which agreement is obtained early. In particular, if agreement cannot be obtained before time $t+1$ (this would happen exactly if $\hat{V}(\rho,\ell)$ is empty for $\ell \leq t$), 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 *weak* protocol for (exact) majority *does* exist: A protocol that either decides that there was a *failure* or decides on the true majority value.

Interestingly, since messages from faulty processors can convey new information about the failure pattern, such messages *do* affect the construction. Therefore, a faulty processor can play an important role in determining what facts become common knowledge and when, even after the processor has been discovered to be faulty. In the crash failure model, however, a failed processor does not communicate with other processors after its failing round and has little affect 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 that all of the analysis in this section carries over into the crash failure model without change. In particular, our construction can be used to derive efficient optimal protocols for simultaneous choice problems in the crash failure model, thus slightly extending [DM]. Ruben Michel has independently characterized the similarity graph in the crash failure model and some of its variants (cf. [Mi]).

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

Proposition 16 : In the omissions model, $C_{\mathcal{N}}\varphi \equiv C_P\varphi$.

This implies that, in the omissions model, simultaneous actions can be performed by *all* processors whenever they can be performed by the set of nonfaulty processors.

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 $\hat{\mathcal{V}}$, 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 $X \equiv \varphi \wedge E_{\mathcal{N}}X$.

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 very similar, they are actually quite different.

For example, Proposition 16 shows that in the omissions model nonvalid facts do become common knowledge to the set P of all processors. In contrast, the following proposition tells us that the only facts that are common knowledge to the group of all processors at a point (ρ, ℓ) in the receiving omissions model are facts valid at time ℓ, where a fact φ is said to be valid (in the system) at time ℓ if "$time = \ell \supset \varphi$" is valid in the system.

Proposition 17: Let $t \geq 2$. In the receiving omissions model, $(\rho, \ell) \models C_P\varphi$ iff φ is valid at time ℓ.

Consequently, no interesting simultaneous action can ever be guaranteed to be performed in this model by the set of all processors, even if all communication is successful.

In the receiving omissions model, a nonfaulty processor receives all of the messages required by the protocol to be sent to it. Thus, at time $k+1$ all nonfaulty processors have an identical view of the first k rounds.

Theorem 18: Let φ be a fact about the first k rounds and let ρ be a run of $\mathcal{F}$ in the receiving omissions model. Then $(\rho, k) \models \varphi$ iff $(\rho, k+1) \models C_{\mathcal{N}}\varphi$.

Theorem 18 implies that given a practical simultaneous choice $\mathcal{C}$, $\mathcal{P}_c$ can be efficiently implemented and can perform actions one round after the first condition *enabled*(a_i) is determined to hold. In particular, simultaneous Byzantine agreement can be obtained in one round in this model.

These results show that while at first glance the assignment of responsibility for undelivered messages to sending or to receiving processors may seem arbitrary, the assignment dramatically affects what facts are common knowledge, and hence when simultaneous actions can be performed. We consider this to be an indication that the omissions model is not a robust failure model.

6.3 Generalized Omissions

Perhaps a more natural failure model is the generalized omissions model, in which faulty processors might omit both to send *and* to receive messages. We first consider generalized omissions with information, a model in which a processor that does not receive a message can determine whether it or the sender is at fault. In this case, the construction used for the omissions model can be modified to yield a similar set of views $\overline{V}(\rho, \ell)$ that are common knowledge at (ρ, ℓ). This yields a similar method of deriving efficient optimal protocols for simultaneous choice problems in this model. (Roughly speaking, the main difference between the two constructions is that receiving failures in round k count as sending failures in round $k+1$. See the full paper for details.) Thus, efficient compiled protocols are possible in this model:

Theorem 19: Let $\mathcal{C}$ be a practical, implementable simultaneous choice. In the generalized omissions model with information, $\mathcal{P}_{\mathcal{C}}$ can be compiled into a polynomial-time, standard protocol for $\mathcal{C}$.

In general, however, such failure information is not available. In the generalized omissions model, an undelivered message implies *only* that either the sender or the intended receiver is faulty. Recall that a processor's view depends only on the labeled communication graph of the run. In all more benevolent failure models a missing edge in this graph uniquely identifies a faulty processor. However, now undelivered messages merely place constraints on which sets of processors could be faulty. There can be many different ways to ascribe failures to processors in a manner consistent with this graph.

The following theorem shows that the computational complexity of determining what facts are common knowledge is dramatically greater with generalized omissions than with the more benign failure models. This theorem holds, however, only for nontrivial facts. We say that a practical fact φ is ***nontrivial*** if it is undetermined (i.e., neither φ nor $\neg\varphi$ is valid in the system) and if it is possible to compute in polynomial-time a labeled communication graph $\mathcal{G}(\rho, \ell)$ that determines that ρ satisfies φ. We say that a practical simultaneous choice is nontrivial if each condition *enabled*(a_i) is nontrivial. Notice that any natural simultaneous choice is nontrivial. The following theorem shows that the problem of determining whether such nontrivial practical facts are common knowledge is co-NP hard.

Lemma 20: Let $\varphi_1, \ldots, \varphi_m$ be nontrivial practical facts. In the generalized omissions model, the problem of determining whether $(\rho, \ell) \models \bigvee_i C_{\mathcal{N}}\varphi_i$ is co-NP hard.

As an immediate corollary, we have:

Theorem 21: In the generalized omissions model, any optimal protocol for a nontrivial practical simultaneous choice requires processors to be able to perform co-NP hard tasks.

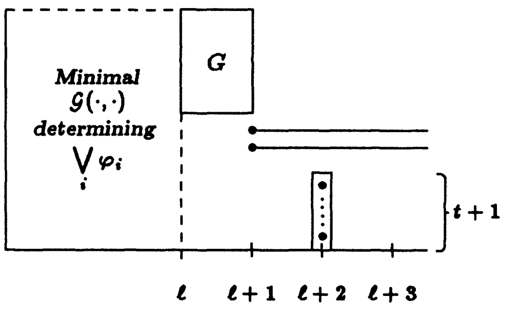

Figure 5: Embedding a graph G in a run ρ.

It follows that optimal protocols for simultaneous choice problems as simple as the distributed firing squad problem or simultaneous Byzantine agreement are computationally infeasible, assuming P $\neq$ NP. This is the first known evidence for this being the case.

Lemma 20 is proved by reducing a co-NP complete problem to the problem of determining whether a nontrivial practical fact $\bigvee_i \varphi_i$ holds. We now introduce the co-NP complete problem used in this reduction. We say that a graph G is ***k-coverable*** if G has a vertex cover of size k.

Lemma 22 (Goldreich, Moses, Tuttle): Given (G, k) such that G is a k-coverable graph, determining whether G is not $(k-1)$-coverable is co-NP complete.

Given a k-coverable graph G, we construct a run ρ as illustrated in Figure 5. $\mathcal{G}(\cdot,\cdot)$ is a minimal labeled communication graph determining that ρ satisfies $\bigvee_i \varphi_i$. In ρ, part of the communication graph during round $\ell+1$ is an embedding of G in which an edge from node v to node w in G is represented by an undelivered message from processor p_v to p_w. In addition, two processors are silent from time $\ell+1$ on, and $t+1$ processors do not fail in ρ. We then show that G is not $(k-1)$-coverable iff $\bigvee_i C_{\mathcal{N}}\varphi_i$ holds at (ρ, ℓ). (Complete details are given in the full paper.)

In addition to increasing the difficulty of determining *whether* a fact is common knowledge at a point, the following theorem show that the uncertainty about the failure pattern has interesting effects on *when* facts become common knowledge. For example, now for the first time the relationship between the number of processors n and the number of faulty processors t affects when facts become common knowledge, as the following theorem shows.

Theorem 23: In the generalized omissions model:

a) If $n \leq 2t$ then the only facts that are common knowledge at time 2 are facts valid at time 2.

b) If $n > 2t$ then some facts not valid at time 2 do become common knowledge at time 2.

Consequently, when $n \leq 2t$ no nontrivial simultaneous choice can be performed at time 2 in this model. We remark that this is the first evidence of behavior in a benevolent failure model depending on the ratio of n and t. Theorem 23 can be used to show that protocols optimal in the generalized omissions model will *not* be optimal in the omissions model.

7 Conclusions

This paper applies the theory of knowledge in distributed systems to thoroughly analyze the design 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 prove 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, it would be interesting to know whether a similar 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 is quite very different from performing simultaneous actions (cf. [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* (cf. [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 $t > 1$, no single protocol can guarantee to reach agreement 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 behave correctly (see [BL] and [CDDS] for definitions of similar problems that *are* implementable in the more malicious models).

We have shown how to derive optimal protocols for nontrivial simultaneous choice problems in the generalized omissions model, requiring processors to perform PSPACE computations between consecutive rounds. We have also shown that any optimal protocol for such a problem must require the processors to perform co-NP hard computations between rounds. Determining the precise complexity of this task is a non-trivial 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 (cf. [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 co-NP hardness result from the generalized omissions model should extend to these models. Furthermore, capturing the precise combinatorial structure of the similarity graph in these models is bound to expose many of the mysterious properties of the 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 co-NP hard computations between consecutive rounds of communication, it is natural to ask what is the earliest time that such actions can be performed by *resource bounded* processors (e.g., processors that can only perform 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 (cf. [CM], [DM], [FI], [HM], and [PR]), is not appropriate for reasoning about such questions. A major challenge motivated by this is the formulation of useful theories 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 the proof of Lemma 22. We thank Oded Goldreich, Joe Halpern, Amos Israeli, and Moshe Vardi for comments that improved the presentation of this work, and we thank Paul Beame, Cynthia Dwork, Vassos Hadzilacos, and David Peleg for stimulating discussions on the topic of this paper.

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 1985.

[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.

[H] 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.

[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.

[MT] Y. Moses and M. Tuttle, Programming simultaneous actions using common knowledge, *MIT Technical Report, MIT/LCS/TM-312*.

[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.