A Link Between Knowledge and Communication in Faulty Distributed Systems
(Preliminary Report)

Murray S. Mazer
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 1A4[1]

ABSTRACT

We identify new circumstances under which processes in faulty distributed systems must communicate for one process to gain knowledge about another. Our main result says that, in systems with process crash failures, message loss, or asynchronous processes, if a proposition of a certain type about a process p does not hold, and later another process q knows that the proposition holds, then there was a message chain from p to q. Systems in which processes vote, bid, or transmit private values are often ones in which processes gain knowledge of propositions of the sort described in our result. One can use this result as a new tool in showing message lower bounds and impossibility results and in designing protocols. We demonstrate this by showing a new impossibility result for commitment problems: if a round-based commitment protocol is resilient to process failures and recovery and such that a message may be received only in the round in which it is sent, then the protocol may run forever.

1 INTRODUCTION

We identify new circumstances under which processes in faulty distributed systems must communicate for one process to gain knowledge about another. This yields a new tool for showing message lower bounds and impossibility results and for designing protocols. The first step is to analyze the specification of a given problem to determine the knowledge each process needs to solve the problem. Then our knowledge gain result prescribes communication between processes so that the processes can gain the required knowledge; this yields the necessary underlying communication structure of any protocol to solve the given problem. One can then use this communication structure for impossibility results and protocol design.

Informally, our result says that, if at some time a proposition ϕ about process p is false and at some later time another process q knows that ϕ is true, then q received a message through some chain of message passing which originated at p, given one of the following conditions:

- processes can crash-fail and ϕ cannot become true while p is failed;
- messages can be lost and ϕ is never forced to become true; or
- processes are asynchronous.

We call this result the *Message Chain Theorem.*

[1]The author's current address is Digital Equipment Corporation, 550 King Street LKG1-2/A19, Littleton, MA 01460, U.S.A. Electronic address: mazer@crl.dec.com.

There are related results in the literature. Halpern and Moses (1990) showed that processes cannot attain common knowledge of certain propositions in systems with unguaranteed communication, asynchronous processes, or asynchronous communication. Chandy and Misra (1986) showed for asynchronous systems that if a process q "gains the knowledge" that another process p knows some proposition, then q must receive a message through some chain of message passing which originates at p. Further, if p later loses its knowledge, then there must be a message chain from q to p. Hadzilacos (1989) showed, in systems with process crash failures or message loss, that a receive event is the only event by which a process may acquire knowledge about certain eventual states of other processes.

Unlike the results of Halpern and Moses (1990), our result addresses knowledge weaker than common knowledge, in systems different from the ones which they considered. We can use the Message Chain Theorem to reason about both finite and infinite knowledge levels and about incremental gains in a process's knowledge through communication. We extend the knowledge gain result of Chandy and Misra (1990) to systems with failures, regardless of process synchrony. Unlike Hadzilacos, we make explicit the first and last processes in the message chain, and we consider more general conditions.

We have used our knowledge gain result extensively for lower bounds, impossibility results, and protocol design for the problem of negotiated commitment [Maze89], a problem closely related but incomparable to atomic commitment. We believe, however, that the result is more generally applicable; systems in which processes vote, bid, or transmit private values are often ones in which processes gain knowledge of propositions of the sort described in the Message Chain Theorem. Furthermore, although some of the impossibility results already have combinatorial proofs (at least for atomic commitment) which depend upon the communication inherent in the problem solution, the knowledge-theoretic proofs using message chains give semantic reasons, based on the problem specification, for the inherent communication. That is, a knowledge analysis of a problem determines, from the specification, the knowledge each process needs to solve the problem, without reference to particular protocols. The Message Chain Theorem provides a tool to link directly the communication required of any protocol for the problem to the knowledge gains needed to solve the problem. One cannot always easily determine how a combinatorial proof reflects the problem being solved.

The paper proceeds as follows. We define, in Sections 2 and 3 respectively, the types of distributed systems we address and the knowledge of processes in such systems. Then, in Section 4, we give the Message Chain Theorem. In Section 5, we define the negotiated commitment problem, followed in Section 6 by a new impossibility result about commitment protocols, based on a knowledge analysis of the problem specification and the resultant communication requirements.

2 MODEL OF DISTRIBUTED SYSTEMS

2.1 Executions

Adapting [Hadz89], we consider a distributed system to consist of two types of elements: ① processes, which execute events (let Π represent the set of n processes); and ② a communication system, $\mathcal{N}$, which contains a set of message packets (of the form $\langle p, \underline{m}, q, i \rangle$, representing the message $\underline{m}$ sent from p to q at time i). The events are of two kinds: communicative and noncommunicative. The communicative events are $\mathsf{SEND}(\underline{m}, q)$ (the executing process sends message $\underline{m}$ to process q, where $\underline{m} \in \underline{M}$, a message vocabulary) and $\mathsf{RECV}(\underline{m}, q)$ (the executing

process receives message $\underline{m}$ from process q). $\underline{m}$ may be the null message λ or a message from $\underline{M}$. These are the only two events by which a process may communicate externally; all other events are local and have no effect on the communication system.

A possible joint behaviour over time of the processes and the communication system is modelled by an *execution* (or *run*). Each execution e is a function mapping time to a global state tuple of the form $\langle time, history_{p1}, history_{p2}, \ldots, history_{pn}, packets \rangle$. *time* represents the time at which the system is observed; $history_{p_i}$ represents the finite sequence of events executed by process p_i in execution e up to the observation instant; and *packets* is the set of message packets in transit at that instant. The *points* of an execution set $\mathcal{E}$, $\mathbf{Pts}(\mathcal{E})$, are $\{(e, f) \mid e \in \mathcal{E}$ and $f \in N\}$.

For $p \in \Pi$, we write $e(f, p)$ for $history_p$, p's history element in the tuple at point (e, f); similarly, we write $e(f, \mathcal{N})$ for *packets*, the set of message packets in the communication system at point (e, f). $d \dashv e(f, p)$ asserts that d is the last event in the sequence $history_p$ at point (e, f); $d \in e(f, p)$ indicates that event d appears in the sequence; $| e(f, p) |$ indicates the number of events in the sequence; and $e(f, p) \cdot d$ indicates the concatenation of event d to the sequence. We write $d \sqsubset (e, f+1, p)$ to say that process p has just executed event d at point $(e, f+1)$, i.e., $d \sqsubset (e, f+1, p)$ iff $e(f+1, p) = e(f, p) \cdot d$. $e(f+1, p) \geq e(f, p)$ denotes that p's event sequence up to time $f+1$ in e has, as a prefix, p's event sequence up to f in e.

For each $p \in \Pi$, we define a relation $=_p$ on the points in system $\mathcal{E}$ which captures when p has the same event sequence in two points. For $(e, f), (e', g) \in \mathbf{Pts}(\mathcal{E})$, we write $(e, f) =_p (e', g)$ iff $e(f, p) = e'(g, p)$. For process set $P \subseteq \Pi$, $(e, f) =_P (e', g)$ iff $(e, f) =_p (e', g)$ for all $p \in P$. Similarly, the communication system is the same in both points, written $(e, f) =_{\mathcal{N}} (e', g)$, iff $e(f, \mathcal{N}) = e'(g, \mathcal{N})$.

We define an *historical equivalence* relation on points as follows: given two executions $e', e \in \mathcal{E}$ and instant $f \in N$, $(e', f) \equiv (e, f)$ iff, for all $0 \leq g \leq f$, $(e', g) =_\Pi (e, g)$ and $(e', g) =_{\mathcal{N}} (e, g)$. This says that two points are historically equivalent if the two executions are indistinguishable over all preceding and current observation instants. A point (e', g) *extends* (e, f), written $(e', g) \geq (e, f)$, iff $(e', f) \equiv (e, f)$ and $g \geq f$. An execution e' *extends* a point (e, f) iff $(e', f) \equiv (e, f)$.

Executions conform to the following informal operational behaviour. At the beginning of time, time 0, the communication system is empty, and no process has executed any events. Each process executes at most one event between successive observation instants. A message is removed from the communication system if it is received or lost. Only messages which were sent but not yet removed may appear in the communication system. We describe this behaviour axiomatically in [Maze89].

2.2 Systems of Executions

Informally speaking, we characterize the behaviours of a distributed protocol by a set of executions $\mathcal{E}$ over Π and $\mathcal{N}$. An execution set must exhibit some closure properties which ensure that if the set represents certain behaviours, then it represents certain other behaviours; these properties capture the ways in which one process's event sequence and the behaviour of the communication system affect another process's event sequence. We call such a closed execution set a *system*. Intuitively, the ability of a process to execute some event should not depend on the events executed so far by other processes or on the behaviour of the communication system, unless the event is a receive — a process can execute a RECV event only if there is an appropriate message in the communication system, and such a message must have been sent

by some process.

For $P, Q \subseteq \Pi$, $e(f,\mathcal{N})[P,Q] \stackrel{\text{def}}{=} \{\langle p, \underline{m}, q, i\rangle \in e(f,\mathcal{N}) \mid p \in P \text{ and } q \in Q\}$. That is, $e(f,\mathcal{N})[P,Q]$ is the set of messages in transit from processes in P to processes in Q at instant f of execution e. $\overline{P} \stackrel{\text{def}}{=} \Pi \setminus P$.

Given a set of executions $\mathcal{E}$, points $(e',g),(e,f) \in \mathbf{Pts}(\mathcal{E})$, and process set $P \subseteq \Pi$, $\mathbf{replace}((e',g),P,(e,f))$ is the set of executions $e'' \in \mathcal{E}$ which extend (e',g) such that
① in e'', each member of $\overline{P}$ executes the same event at $(e'',g+1)$ as it did at $(e',g+1)$,
② in e'', the event executed by each member of P at $(e',g+1)$ is *replaced* by the event it executed at $(e,f+1)$, and
③ the messages from P to Π are the messages not sent or received in e' between g and $g+1$, plus any messages received in e' at $g+1$ by $p \in P$ but not in e by p at $f+1$, plus any messages newly sent by P; and the messages from $\overline{P}$ to Π are whatever was in e' at $g+1$, plus whatever P received in e' at $g+1$, minus whatever P received in e at $f+1$ or was lost in transit to P in e at $f+1$ [Hadz89].

The progress-closure properties we require here are these:[2]

S1 (Nonreceive Progress): The ability of a process to perform a nonRECV event depends on the process's behaviour only.

Let $(e,f),(e',g) \in \mathbf{Pts}(\mathcal{E})$ and $p \in \Pi$ be such that

- $(e,f) =_p (e',g)$,
- $\mathsf{RECV}(\underline{m},q) \not\sqsubset (e,f+1,p)$, for any message $\underline{m}$, $q \in \Pi$

Then $\mathbf{replace}((e',g),\{p\},(e,f)) \neq \emptyset$.

Sλ (Null Receive Progress): A process' ability to receive a null message from another process depends on the state of the former process, the messages sent to the recipient by the latter process, and the behaviour of the communication system.

Let $e, e' \in \mathcal{E}$, $g \in N$, and $q,p \in \Pi$ be such that

- $(e,g) =_p (e',g)$,
- $e'(g,\mathcal{N})[\{q\},\{p\}] \subseteq e(g,\mathcal{N})[\{q\},\{p\}]$,
  (There are no different messages in the communication system from q to p up to time g in e' than in e), and
- $\mathsf{RECV}(\lambda,q) \sqsubset (e,g+1,p)$.

Then $\mathbf{replace}((e',g),\{p\},(e,g)) \neq \emptyset$.

S3 (Available Receive Progress): Once a process p has sent a message to another process q, q's ability to receive the message cannot depend on p's subsequent behaviour; q's ability to receive the message does depend on the communication system.

Let $e, e' \in \mathcal{E}$, $f,g \in N$, and $q,p \in \Pi$ be such that

- $(e,g) =_p (e',g)$
- $\mathsf{RECV}(\underline{m},q) \sqsubset (e,g+1,p)$,

[2] These closure properties correspond to a natural set of assumptions on the protocols which "produce" the behaviours in the execution sets [Maze89].

- $\langle q, \underline{\mathrm{m}}, p, f\rangle \in e'(g, \mathcal{N})$ (the message is still available in e' at g, as in e)
- let $\eta = \{\langle q, \underline{\mathrm{m}}, p, i\rangle \mid \langle q, \underline{\mathrm{m}}, p, i\rangle \in e'(g, \mathcal{N})$ and $0 \leq i \leq f\}$;
  $\eta \subseteq e(g, \mathcal{N})[\{q\}, \{p\}]$ (The set of messages available at (e', g), sent before $f+1$, and from q to p are a subset of those available at (e, g).)

Then $\mathbf{replace}((e', g), \{p\}, (e, g)) \neq \emptyset$.

2.3 System Characteristics

A system $\mathcal{E}$ is *weakly terminating* if every point of $\mathcal{E}$ can be extended to a point beyond which no process executes any more events in any extension [KoTo88].

Definition 1 A system $\mathcal{E}$ is *weakly terminating* if, for each point $(e, f) \in \mathbf{Pts}(\mathcal{E})$, there is (e', g) extending (e, f) such that, for all $p \in \Pi$ and all $(e'', h) \in \mathbf{Pts}(\mathcal{E})$ extending (e', g), $e''(h, p) = e'(g, p)$.

(e', g) is a *terminating extension* of (e, f) and a *terminating point* of $\mathcal{E}$. ⊠

We capture process crash failures and message loss by another set of closure properties. Informally, a system is *subject to process failures* if any process subset may fail at any time. A failure of process p is modelled by a FAIL event in an event sequence for p. We collect into $\mathbf{fail}(e, f, P)$ each execution e' in an execution set $\mathcal{E}$ such that ① e' extends $(e, f-1)$; ② processes other than those in P execute the same events at (e', f) as at (e, f); ③ each nonterminated member of P is failed; and ④ any message that $p \in P$ sends at (e, f) does not appear in $e'(f, \mathcal{N})$, and any message that $p \in P$ receives at (e, f) appears in $e'(f, \mathcal{N})$.

Definition 2 A system $\mathcal{E}$ is *subject to process failures* if, for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$:
(any process subset may fail) for any $P \subseteq \Pi$, $\mathbf{fail}(e, f, P) \neq \emptyset$. ⊠

Let *Failed*(e, f) represent the set of failed processes at point (e, f):

$$\textit{Failed}(e, f) = \{p \mid p \in \Pi \text{ and } \mathsf{FAIL} \dashv e(f, p)\}.$$

A system is *subject to process failures and recovery* if, at any time, any process subset may fail, and any subset of failed processes may recover.

Definition 3 A system $\mathcal{E}$ is *subject to process failures and recovery* if, for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$:

(any process subset may fail) for any $P \subseteq \Pi$, $\mathbf{fail}(e, f, P) \neq \emptyset$.

(any subset of failed processes may recover) for each nonempty $P \subseteq \textit{Failed}(e, f)$, there is $(e', g) \in \mathbf{Pts}(\mathcal{E})$ properly extending (e, f) such that $\mathsf{FAIL} \not\dashv e'(g, p)$, for all $p \in P$. ⊠

A system is *subject to communication failures* if any subset of messages in transit at any time may be lost. Let M be a subset of the messages in transit at point (e, f): $M \subseteq e(f, \mathcal{N})$. We collect into the set $\mathbf{lose}(e, f, M)$ the set of executions $e' \in \mathcal{E}$ such that e' extends $(e, f-1)$, $(e', f) =_\Pi (e, f)$, and $e'(f, \mathcal{N}) = e(f, \mathcal{N}) \setminus M$.

Definition 4 A system $\mathcal{E}$ is *subject to communication failures* if, for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$ and any $M \subseteq e(f, \mathcal{N})$: $\mathbf{lose}(e, f, M) \neq \emptyset$. ⊠

We also model systems in which a message has a maximum lifetime in transit. A system is *k-transit bounded* if any message sent disappears from the communication system at most k time units after being sent.

Definition 5 A system $\mathcal{E}$ is *k-transit bounded* (for some finite $k \geq 0$) if, for any $\underline{m} \in \underline{M}$, $q,p \in \Pi$, $e \in \mathcal{E}$: if $\mathsf{SEND}(\underline{m}, q) \sqsubset (e, f, p)$, then $\langle p, \underline{m}, q, f\rangle \notin e(f+k, \mathcal{N})$. ⊠

Round-based protocols typically assume k-transit bounded systems.

A system has *asynchronous processes* if, at any point, any process may execute no event in the next time step.

Definition 6 A system $\mathcal{E}$ has *asynchronous processes* if, for any $e \in \mathcal{E}$, $f > 0$, $p \in \Pi$, there is $e' \in \mathcal{E}$ extending $(e, f-1)$ such that

- $(e', f) =_p (e, f-1)$ (p executes no event),
- $(e', f) =_{\Pi \setminus \{p\}} (e, f)$ (the others do as they did in e), and
- $e'(f, \mathcal{N}) = (e(f, \mathcal{N}) \cup A) \setminus B$ (the communication system is adjusted), where A contains the message packet for any message p receives at (e, f) and B contains the message packet for any message p sends at (e, f).

3 KNOWLEDGE LOGIC

We use the knowledge logic of Halpern and Moses (1990), including process knowledge and collective knowledge within process sets. The language of the logic has the following symbols: a set Φ of primitive propositions; a finite set Π of process names; $\{\neg, \vee, \Box, (,)\}$; $\{K_x \mid x \in \Pi\}$; and $\{K_X \mid X \subseteq \Pi, X \neq \emptyset\}$. The set of well-formed formulae (or *wffs*) $\mathcal{L}_\Pi(\Phi)$ is the smallest set such that (1) every member of Φ is a well-formed formula, and (2) if ϕ and ψ are well-formed formulae, then so are $(\neg\phi)$, $(\phi \vee \psi)$, $\Box\phi$, $K_x\phi$, $K_X\phi$. We abbreviate $(\neg((\neg\phi) \vee (\neg\psi)))$ by $(\phi \wedge \psi)$ and $((\neg\phi) \vee \psi)$ by $(\phi \supset \psi)$[3].

We interpret wffs via possible worlds semantics relative to an *interpreted system* (or *model*), a structure $\mathbf{M} = (\mathcal{E}, \mathcal{A})$ in which $\mathcal{E}$ is a system over Π and $\mathcal{N}$, and $\mathcal{A}{:}\Phi \rightarrow 2^{\mathbf{Pts}(\mathcal{E})}$ is an interpretation mapping each primitive proposition to the set of points in $\mathcal{E}$ in which the proposition holds. The points of the system are the possible worlds. Knowledge is based on a complete history interpretation [HaMo90].

Given a model $\mathbf{M}$, we write $(\mathbf{M}, e, f) \models \phi$ to express that wff ϕ holds in point (e, f) of the model. (If $\mathbf{M}$ is understood from context, we write $(e, f) \models \phi$.) We define $\models$ as follows (assume $\phi, \psi \in \mathcal{L}_\Pi(\Phi)$):

[Primitives] For $\phi \in \Phi$, $(\mathbf{M}, e, f) \models \phi$ iff $(e, f) \in \mathcal{A}(\phi)$.

[Negation] $(\mathbf{M}, e, f) \models (\neg\phi)$ iff $(\mathbf{M}, e, f) \models \phi$ does not hold.

[Disjunction] $(\mathbf{M}, e, f) \models (\phi \vee \psi)$ iff $(\mathbf{M}, e, f) \models \phi$ or $(\mathbf{M}, e, f) \models \psi$ (inclusively).

[Henceforth] $(\mathbf{M}, e, f) \models \Box\phi$ iff, for all $e' \in \mathcal{E}$ such that $(e, f) \equiv (e', f)$, $(\mathbf{M}, e', g) \models \phi$ for all $g \geq f$. ("henceforth ϕ" holds in point (e, f) iff ϕ holds now and in any possible extension of (e, f).)

[Process Knowledge] For $p \in \Pi$, $(\mathbf{M}, e, f) \models K_p\phi$ iff $(\mathbf{M}, e', g) \models \phi$, for all $(e', g) \in \mathbf{Pts}(\mathcal{E})$ such that $(e, f) =_p (e', g)$. ("p knows ϕ" iff ϕ is true in all points which look to p similar to the current one.)

[Collective Knowledge] For $P \subseteq \Pi$, $(\mathbf{M}, e, f) \models K_P\phi$ iff $(\mathbf{M}, e', g) \models \phi$,

[3] In the sequel, we elide the parentheses "(" and ")" in the usual way in formulae in which no ambiguity results. Furthermore, for clarity, we sometimes use "[" for "(" and "]" for ")".

for all $(e', g) \in \mathbf{Pts}(\mathcal{E})$ such that $(e, f) =_P (e', g)$. ("the members of P collectively know ϕ" iff ϕ holds in all points which the members of P collectively think possible.)

A wff ϕ is *valid in structure* $\mathbf{M}$, written $\mathbf{M} \models \phi$, iff $(\mathbf{M}, e, f) \models \phi$ for all points $(e, f) \in \mathbf{Pts}(\mathcal{E})$. Processes in this logic have the *introspection* property: for any wff ϕ, $P \subseteq \Pi$, and model $\mathbf{M}$, $\mathbf{M} \models K_P\phi \supset K_P K_P \phi$ and $\mathbf{M} \models \neg K_P\phi \supset K_P \neg K_P \phi$.

We now identify a set of useful and important properties of formulae in interpreted systems. Then we relate these concepts to each other.

Stable: A wff ϕ is *stable* (in $\mathbf{M}$) if the following property holds: $\mathbf{M} \models \phi \supset \Box\phi$. A stable wff stays true forever after it becomes true [ChLa85]. Stability is useful for expressing immutable choices and decisions, such as the choice to accept a contract or commit a transaction.

Local: A formula ϕ is *local to* P (in $\mathbf{M}$), for $P \subseteq \Pi$, if $\mathbf{M} \models K_P\phi \vee K_P\neg\phi$. That is, P always knows the truth value of ϕ [ChMi86]. Local formulae are intended to model predicates whose value is controlled by or locally testable by the actions of the processes to which the formulae are local. If $P = \{p\}$, we write that ϕ is local to p instead of $\{p\}$.

P-failure-dissociated: A formula ϕ is called *P-failure-dissociated* (in $\mathbf{M}$), for $P \subseteq \Pi$, if, whenever ϕ is false, ϕ remains false as long as a process in P is failed [Hadz89]. That is, for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$, if $(e, f) \models \neg\phi$ and $\mathsf{FAIL} \dashv e(f+1, p)$, for any $p \in P$, then $(e, f+1) \models \neg\phi$. If $P = \{p\}$, we write that ϕ is p-failure-dissociated instead of $\{p\}$-failure-dissociated.

P-receive-dependent: A formula ϕ is called *P-receive-dependent* (in $\mathbf{M}$) if, when it is false, it can become true only if some process in P receives a nonnull message from a process not in P [Hadz89]. That is, for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$, if $(e, f) \models \neg\phi$ and $(e, f+1) \models \phi$, then $\mathsf{RECV}(\underline{m}, q) \sqsubset (e, f+1, p)$, for some $p \in P$, $\underline{m} \neq \lambda$, and $q \in \overline{P}$. If $P = \{p\}$, we write that ϕ is p-receive-dependent instead of $\{p\}$-receive-dependent.

Pointwise nontrivial: A formula ϕ is called *pointwise nontrivial* (in $\mathbf{M}$) iff, whenever it is false, it may remain false in the next time instant; i.e., for any $(e, f) \in \mathbf{Pts}(\mathcal{E})$, if $(e, f) \models \neg\phi$, then there is $e' \in \mathcal{E}$ such that e' extends (e, f) and $(e', f+1) \models \neg\phi$.

Intuitively, locality means that the proposition is "about" the process set to which the proposition is local. For example, if we were modelling the outcome of a coin toss by process p and wff ϕ represents the proposition "p has flipped a heads", then we expect ϕ to be local to p. Furthermore, we do not expect ϕ to become true while p is failed, so ϕ is p-failure-dissociated. Finally, we expect the outcome of p's coin toss to be fair and not forced to be either heads or tails, so ϕ is pointwise nontrivial.

As the following lemma shows, these concepts are strongly related.

Lemma 7 Given any model $\mathbf{M}$, wff ϕ, and $P \subseteq \Pi$, ϕ is pointwise nontrivial in $\mathbf{M}$ if any of the following holds:

- $\mathbf{M}$ is subject to process failures and ϕ is P-failure-dissociated.
- $\mathbf{M}$ is subject to communication failures and ϕ is P-receive-dependent.
- $\mathbf{M}$ has asynchronous processes and ϕ is local to P.

Further, if $\mathbf{M}$ is subject to process failures and ϕ is P-receive-dependent, then ϕ is P-failure-dissociated. □

4 MESSAGE CHAINS

Given an execution e, a *message chain from process p to process q in interval (e,f) to (e,g)* is a sequence of send/receive pairs such that ($f \leq f_1$; $f_i < f_{i+1}$, for $1 \leq i < 2n$; $f_{2n} \leq g$):
$\mathsf{SEND}(\underline{m}_1, p_1) \sqsubset (e, f_1, p)$; $\mathsf{RECV}(\underline{m}_1, p) \sqsubset (e, f_2, p_1)$; $\mathsf{SEND}(\underline{m}_2, p_2) \sqsubset (e, f_3, p_1)$;
$\mathsf{RECV}(\underline{m}_2, p_1) \sqsubset (e, f_4, p_2)$; ... $\mathsf{SEND}(\underline{m}_n, q) \sqsubset (e, f_{2n-1}, p_{n-1})$; $\mathsf{RECV}(\underline{m}_n, p_{n-1}) \sqsubset (e, f_{2n}, q)$.

The abbreviation $P \xrightarrow{+} Q$ indicates a message chain of length at least one from P to Q (execution and interval will be clear from context). We abbreviate $\{p\} \xrightarrow{+} \{q\}$ as $p \xrightarrow{+} q$.

For any model $\mathbf{M}$, $e \in \mathcal{E}$, nonempty process sets $P \subset \Pi$ and $Q \subset \Pi$ such that $P \cap Q = \emptyset$, and wff ϕ local to P, a *ϕ-message chain* from P to Q in interval (e,f) to (e,i) is a message chain from some $p \in P$ to some $q \in Q$ in an interval (e,j) to (e,i) such that $f < j$, $(\mathbf{M}, e, j-1) \models \neg\phi$, $(\mathbf{M}, e, j) \models \phi$, and $(\mathbf{M}, e, i) \models K_Q\phi$.

Finally, we can give our knowledge gain result. Recall that, informally, our result says that, if at some time a proposition ϕ about process p is false and at some later time another process q knows that ϕ is true, then q received a message through some chain of message passing which originated at p, given one of the following conditions: processes can crash-fail and ϕ cannot become true while p is failed; messages can be lost and ϕ is never forced to become true; or processes are asynchronous.

Theorem 8 (The Message Chain Theorem)
Assume any model $\mathbf{M}$, nonempty $P \subset \Pi$ and $Q \subset \Pi$ such that $P \cap Q = \emptyset$, wff ϕ local to P, point (e,f) in $\mathcal{E}$, and $i > f$ such that

- $(\mathbf{M}, e, f) \models \neg\phi$ and $(\mathbf{M}, e, i) \models K_Q\phi$, and
- one of
  - $\mathbf{M}$ is subject to process failures and ϕ is P-failure-dissociated in $\mathbf{M}$, or
  - $\mathbf{M}$ is subject to communication failures and ϕ is pointwise nontrivial in $\mathbf{M}$, or
  - $\mathbf{M}$ has asynchronous processes.

There is a ϕ-message chain from P to Q in (e,f) to (e,i). □

We can give an intuitive explanation of the Message Chain Theorem in terms of potential causality between events, using Lamport's "happened before", or *affects*, relation (Lamport 1978). One way to view the claim in the Message Chain Theorem is that there must be ① an event d_p in the event sequence of some process in P between times j (at which ϕ starts to hold) and i in e, and ② an event d_q in the event sequence of some process in Q in the same interval, such that d_p affects d_q. Because $p \neq q$, there must be a message chain between the two processes. If no such events exist, then the events of P in the interval of interest do not affect the events of Q in that interval, so there is another execution possible in which the processes in Q do as in (e,j) to (e,i) and those in P execute other events which ensure that ϕ is false when q comes to "know" ϕ, yielding a contradiction.

The proof of this result is long and detailed; see [Maze89]. The proof follows a common structure for each of the three conditions, to wit:

1. Use pointwise nontriviality to identify an execution $e' \in \mathcal{E}$ which extends $(e, j-1)$ such that $(e', j) \models \neg\phi$.

2. Identify $e'' \in \mathcal{E}$ extending $(e', j-1)$ such that $(e'', i) =_Q (e, i)$ and $(e'', i) \models \neg\phi$, which yields the contradiction $(e'', i) \models \phi \wedge \neg\phi$. Do this by inductively identifying a sequence of intermediate executions $e^0, e^1, \ldots, e^k$, for $k = i - j$, in which

   (a) Q's event sequence from e is exactly preserved up to time i, by using the progress-closure properties (otherwise, we could not guarantee that $(e'', i) \models \phi$!)

   (b) P's event sequence is *not* preserved (otherwise $(e'', i) \models \phi$!) We invoke the relevant conditions on $\mathbf{M}$ and ϕ for any member of P to prevent any messages from P sent at or after (e', j) from reaching any other process. We do this by having the members of P fail instead of sending such a message, having the message lost when sent, or having the members of P freeze just before sending the message.

   (c) for each $q' \in \Pi \setminus (Q \cup P)$, q''s event sequence from e is maintained via the progress-closure properties as long as possible, to support Q's event sequence (to ensure any message sent by q' to Q, which message is not part of a chain from P to Q in (e, j) to (e, i), can still be received by Q). The event sequence from e for q' is preserved unless q' receives a message which is along a message chain from P beginning at or after (e, j). For any such q', we change its event sequence, again invoking the relevant conditions on $\mathbf{M}$ and ϕ to prevent messages from q' from entering the communication system. Informally speaking, this cannot affect Q's ability to execute the same events as in e, because we assume that there is no chain from P to Q in e, so in particular there is none through q', so there is no chain from q' to Q from the instant q''s sequence changes.

We note that, as a corollary of the Message Chain Theorem, if ϕ in the statement of the theorem is initially false in $\mathbf{M}$, then ϕ requires a ϕ-message chain $P \xrightarrow{+} Q$ to become known to Q in $\mathbf{M}$. Further, if ϕ requires $P \xrightarrow{+} Q$ to become known to Q in $\mathbf{M}$, then $K_Q\phi$ is Q-receive-dependent in $\mathbf{M}$. Finally, using the asynchronous processes case of the Message Chain Theorem, one can easily show the knowledge gain and loss results of [ChMi86] mentioned in Section 1 [Maze89].

5 NEGOTIATED COMMITMENT

We now introduce the negotiated commitment problem and illustrate the use of the Message Chain Theorem, by analyzing negotiated commitment. The problem of negotiated commitment is to ensure that the processes in a distributed negotiation commit consistently to the outcome, even in the face of failures. Negotiated commitment is related, but incomparable, to the problem of atomic commitment in distributed transaction systems, differing as follows. First, in negotiated commitment, a distinguished process, historically called the *manager*, coordinates the commitment; in atomic commitment, there need not be a single coordinator. Second, in negotiated commitment, commitment may be established among subsets of the participating processes; in atomic commitment, a commitment must include all processes. The specifications of these two problems reflect these differences. Our lower bound and impossibility results for negotiated commitment convert easily to apply to atomic commitment.

For negotiated commitment, we divide the processes in the system into two disjoint sets: the manager, $\{m\}$, and the *contractors*, or bidders, $\mathcal{C}$. Each of the contractors chooses whether to bid or not on an announced contract. The manager selects from among the bidding contractors

to establish a *dependency set*, representing those contractors which the manager wants to commit to performing the announced task; contractors not in the dependency set must not carry out the task. We represent contractor c's choice to bid by a primitive proposition BID_c; we represent its choice not to bid by $NO\text{-}BID_c$. We represent the manager's possible dependency set choices by the primitive propositions $DEPEND_m^x$ for each nonempty $x \subseteq \mathcal{C}$. For each $c \in \mathcal{C}$, we define the allowed dependencies set $\mathcal{D}_c \subseteq \{x \mid x \in 2^{\mathcal{C}} \text{ and } c \in x\}$. The manager records locally a decision outcome for each contractor, either $AWARD_m^c$, representing that m expects c to carry out the task, or $REJECT_m^c$, representing that m expects c not to carry out the task. Similarly, each contractor records locally a decision outcome, either $ACCEPT_c$, representing that c will carry out the contract, or $REFUSE_c$, that c will not carry out the contract. Informally, the processes reach consistent commitment if, for each $c \in \mathcal{C}$, the manager decides $AWARD_m^c$ and c decides $ACCEPT_c$, or m decides $REJECT_m^c$ and c decides $REFUSE_c$. Each BID_c, $NO\text{-}BID_c$, $ACCEPT_c$, and $REFUSE_c$ proposition is stable and local to c; BID_c is also c-failure-dissociated. Each $DEPEND_m^x$, $AWARD_m^c$, and $REJECT_m^c$ proposition is stable and local to m; $DEPEND_m^x$ is also m-failure-dissociated. All of these propositions are initially false.

A *C-system* is an interpreted system which satisfies the specification of negotiated commitment under process or communication failures. The potential for failures in the system makes consistent negotiated commitment nontrivial to achieve. The C-system properties fit into four general categories: necessity, exclusivity, freedom of choice, and decision completion. Five necessity properties tell which propositions must hold when others hold. Eleven exclusivity properties proscribe two propositions from holding simultaneously. Five nontriviality properties capture the notion that a proposition (which may intuitively represent a choice to be made by a process) is not forced to hold at any time. Two decision completion properties describe situations and combinations in which processes must decide. Below, we list the properties relevant to our discussion.

Dependent Acceptance: For all $c \in \mathcal{C}$, $\mathbf{M} \models ACCEPT_c \supset \vee_{x \in \mathcal{D}_c} DEPEND_m^x$.
(An accepted contractor must be a codependent.)

Dependent Award: For all $c \in \mathcal{C}$, $\mathbf{M} \models AWARD_m^c \supset \vee_{x \in \mathcal{D}_c} DEPEND_m^x$.
(An awarded contractor must be a codependent.)

No Unilateral Dependencies: For all $c \in \mathcal{C}$, $\mathbf{M} \models \vee_{x \in \mathcal{D}_c} DEPEND_m^x \supset BID_c$.
(A codependent must have bid.)

Nonintersecting Dependencies:
For each $c \in \mathcal{C}$, for each pair $x, y \in \mathcal{D}_c$ such that $x \neq y$,
$\mathbf{M} \models \neg(DEPEND_m^x \wedge DEPEND_m^y)$.
(c may be involved in at most one dependency set in any one negotiation.)

Decision Harmony: For all $c \in \mathcal{C}$,

$\mathbf{M} \models \neg(AWARD_m^c \wedge REFUSE_c)$ $\quad$ $\mathbf{M} \models \neg(REJECT_m^c \wedge ACCEPT_c)$
(The manager and each contractor can never decide inconsistently.)

$\mathbf{M} \models \neg(AWARD_m^c \wedge REJECT_m^c)$ $\quad$ $\mathbf{M} \models \neg(ACCEPT_c \wedge REFUSE_c)$.
(Only one of two possible decisions is allowed for each process.)

Four other properties state that if contractors c and d are both in the dependency set x, then m cannot decide for them inconsistently, c and d cannot decide inconsistently with each other,

m cannot award to c while a codependent d refuses, and m cannot reject c while a codependent d accept.

It is straightforward to show the following simple knowledge requirements.

Lemma 9 For any C-system $\mathbf{M}$, $c \in \mathcal{C}$, $p \in \Pi$,

- $\mathbf{M} \models K_p(\vee_{x \in \mathcal{D}_c} DEPEND_m^x) \supset K_p BID_c$.
- $\mathbf{M} \models K_p AWARD_m^c \supset K_p(\vee_{x \in \mathcal{D}_c} DEPEND_m^x)$.
- $\mathbf{M} \models K_p AWARD_m^c \supset K_p BID_c$.
- $\mathbf{M} \models K_p ACCEPT_c \supset K_p(\vee_{x \in \mathcal{D}_c} DEPEND_m^x)$.
- $\mathbf{M} \models K_p ACCEPT_c \supset K_p BID_c$. □

Lemma 9 gives us some of the knowledge a contractor needs to accept or a manager needs to award. We can now use the Message Chain Theorem to derive communication requirements from the knowledge requirements. Using Lemma 9 and the Message Chain Theorem, it is easy to show, for systems with process crash failures, message loss, or asynchronous processes, that, for a contractor c to accept, there must be one message chain from c to m, so that m knows that c bid, and a subsequent message chain from m to c, so that c knows that m selected c as a codependent. This matches our intuition about the problem; the important point is that we are able to formalize and validate that intuition directly. As we shall see in Section 6, the knowledge requirements and the corresponding communication requirements are not always so simple.

We have used the Message Chain Theorem to establish a message lower bound and several impossibility results for negotiated commitment. All but one of these results was known in some form for atomic commitment, but we have shown them using knowledge theory and, in particular, the Message Chain Theorem, sometimes providing a more general result. For example, the message lower bound follows from the argument above that, for each contractor which has accepted, two messages must be sent and received in the system. This is a different, and perhaps more direct, proof strategy than that given combinatorially by Dwork and Skeen (1983) and knowledge-theoretically by Hadzilacos (1989) for atomic commitment.

6 THE IMPOSSIBILITY RESULT

The result we give below states that there is no protocol for negotiated commitment which guarantees weak termination in a system in which processes may fail and recover and in which messages may spend a bounded amount of time in transit. All of the impossibility results have the same form: ① determine the knowledge required for the desired commitment behaviour in the specific type of system; ② determine the communication, in terms of message chains, required to establish that knowledge in any protocol; and ③ argue that the communication is unattainable.

To obtain our result, we will prove that, in weakly terminating C-systems which are k-transit bounded and subject to process failures and recovery, ① an accepting contractor c must have arbitrarily deeply nested knowledge about the manager's knowledge about c's knowledge that m made c a codependent, and ② an awarding manager must have arbitrarily deeply nested knowledge about c's knowledge that m made c a codependent. We show this, in Lemma 16,

by induction on the knowledge nesting level. Then the Message Chain Theorem allows us to show that the required knowledge cannot be gained in finite time (Theorem 18), by showing that arbitrarily many consecutive message chains are needed to gain the required knowledge (Lemma 17). In order to show Lemma 16, we must show ① how m's award to c depends on c's acceptance knowledge and c's acceptance depends on m's award knowledge (Lemma 14), and ② that each of the nested levels of knowledge in Lemma 16 can be attained only by message receipt (Lemma 15). Lemma 14 shows that ① if c must know c-receive-dependent proposition ϕ in order to accept, then an awarding m must know that c knows ϕ, and ② if m must know m-receive-dependent proposition ϕ in order to award, then, in order to accept, c must know that m knows ϕ. Lemma 14 and Lemma 15 allow us to show the interleaved knowledge requirement in Lemma 16, using, as a basis, the fact that an accepting c must know that it is a codependent (Lemma 9).

Lemma 10 and Theorem 11 give some preliminary results about termination and decision. Lemma 10 states that, at any terminating point of a C-system subject to process failures and recovery, all processes have decided and no process is failed.

Lemma 10 In a C-system subject to process failures and recovery, if (e, f) is a terminating point, then $(e, f) \models \wedge_{c \in \mathcal{C}}(ACCEPT_c \vee REFUSE_c) \wedge \wedge_{c \in \mathcal{C}}(AWARD_m^c \vee REJECT_m^c)$ and $Failed(e, f) = \emptyset$. □

Theorem 11 states that, in any weakly terminating system which is k-transit bounded and subject to process failures, every point can be extended to a terminating point without any process receiving any further messages. (Koo and Toueg (1988) showed this for weakly terminating systems subject to communication failures; instead of communication failures, we use the combination of k-transit boundedness and process failures to ensure that messages may disappear without being received.)

Theorem 11 Given a weakly terminating model $\mathbf{M} = (\mathcal{E}, \mathcal{A})$ which is k-transit bounded and subject to process failures, for all $(e, f) \in \mathbf{Pts}(\mathcal{E})$, there is a terminating extension (e', g) such that no process receives a nonnull message after (e', f). □

Lemma 12 states that m's dependency set choices cannot start to hold while m is failed and are not forced to hold.

Lemma 12 For any C-system $\mathbf{M}$ and $c \in \mathcal{C}$, $\vee_{x \in \mathcal{D}_c} DEPEND_m^x$ is m-failure-dissociated and pointwise nontrivial.

Based on the Message Chain Theorem and Lemma 12, Lemma 13 asserts that, if a contractor c knows that m made c a codependent, there must be a message chain from m to c.

Lemma 13 For any C-system $\mathbf{M}$, $c \in \mathcal{C}$, and $(e, f) \in \mathbf{Pts}(\mathcal{E})$, if $(e, f) \models K_c \vee_{x \in \mathcal{D}_c} DEPEND_m^x$, then there is a $\vee_{x \in \mathcal{D}_c} DEPEND_m^x$-message chain $m \xrightarrow{+} c$ in $(e, 0)$ to (e, f). □

Therefore, $K_c \vee_{x \in \mathcal{D}_c} DEPEND_m^x$ is c-receive-dependent.

The first part of Lemma 14 below says intuitively that, in order for m to award to c at some point, m must be sure that c has received enough information to accept. This is because otherwise the system may terminate without any more messages being received (by Theorem 11), so that c cannot gain the knowledge it needs to accept and must therefore refuse (by

Lemma 10), violating the **Decision Harmony** property.[4] The second part of Lemma 14 has the corresponding assertion needed for c to accept.

Lemma 14 In any weakly terminating C-system $\mathbf{M}$ which is k-transit bounded and subject to process failures and recovery, for any $c \in \mathcal{C}$,

- if $K_c\phi$ is c-receive-dependent and $\mathbf{M} \models ACCEPT_c \supset K_c\phi$, then $\mathbf{M} \models AWARD_m^c \supset K_m K_c\phi$.
- if $K_m\phi$ is m-receive-dependent and $\mathbf{M} \models AWARD_m^c \supset K_m\phi$, then $\mathbf{M} \models ACCEPT_c \supset K_c K_m\phi$.

Proof: We prove the first; the second follows analogously. We use the following claim: $\mathbf{M} \models AWARD_m^c \supset K_c\phi$.

Proof of claim: Assume, by way of contradiction, that there is $(e,f) \in \mathbf{Pts}(\mathcal{E})$ such that $(e,f) \models AWARD_m^c \wedge \neg K_c\phi$. Therefore, $(e,f) \models \neg ACCEPT_c$. By Theorem 11, there is $(e',h) \in \mathbf{Pts}(\mathcal{E})$ which is a terminating extension of (e,f) such that no process receives a nonnull message after (e',f). By Lemma 10, $(e',h) \models ACCEPT_c \vee REFUSE_c$, and by $K_c\phi$ being c-receive-dependent, $(e',h) \models REFUSE_c$. Therefore, $(e',h) \models AWARD_m^c \wedge REFUSE_c$, violating **Decision Harmony**. □Claim

Because $AWARD_m^c$ is local to m, whenever $AWARD_m^c$ holds in a particular point, it holds in all points which m considers similar. Therefore, $K_c\phi$ also holds in those points, so $\mathbf{M} \models AWARD_m^c \supset K_m K_c\phi$. □

We will call the first part of Lemma 14 the Award Knowledge Rule and the second part the Accept Knowledge Rule.

Below, we abbreviate $\vee_{x \in \mathcal{D}_c} DEPEND_m^x$ by $DEPEND_m^x$. Lemma 15 shows that, in a C-system subject to process failures, ① c must receive a message for the knowledge level $(K_cK_m)^j K_c DEPEND_m^x$ to hold; and ② m must receive a message for $(K_mK_c)^i DEPEND_m^x$ to hold.[5]

Lemma 15 In a C-system $\mathbf{M}$ subject to process failures, for any $c \in \mathcal{C}$,

- $(K_cK_m)^j K_c DEPEND_m^x$ is c-receive-dependent and initially false, for all $j \geq 0$; and
- $(K_mK_c)^i DEPEND_m^x$ is m-receive-dependent and initially false, for all $i \geq 1$.

Proof: We prove the first by induction on the knowledge nesting level j; the second follows similarly by induction on the knowledge nesting level i.

Base case: $j = 0$. We claim that $K_c DEPEND_m^x$ is c-receive-dependent. This holds from the requirement of a $DEPEND_m^x$-message chain (Lemma 13). $K_c DEPEND_m^x$ is initially false because $DEPEND_m^x$ is so.

---

[4] Unlike agreement problems, in which the post-failure decisions of faulty processes are irrelevant, commitment problems impose the same consistency requirements on all processes, whether they decide before or after failures. This motivates m, when it awards to c, to know that c has received enough information to decide consistently; otherwise, for example, c might fail and not recover until there are no more messages for it to receive, which guarantees that c cannot decide consistently.

[5] For any $p,q \in \Pi$, $j \geq 0$, we abbreviate $\underbrace{K_pK_q}_{1}\underbrace{K_pK_q}_{2}\ldots\underbrace{K_pK_q}_{j}\phi$ by $(K_pK_q)^j\phi$. We also abbreviate $p \xrightarrow{+} q \underbrace{\xrightarrow{+} p \xrightarrow{+} q}_{1}\underbrace{\xrightarrow{+} p \xrightarrow{+} q}_{2}\ldots\underbrace{\xrightarrow{+} p \xrightarrow{+} q}_{j}$ by $p \xrightarrow{+} q(\xrightarrow{+} p \xrightarrow{+} q)^j$.

Inductive step: $j > 0$. Assume the inductive hypothesis holds for $j-1$.
Therefore, $(K_cK_m)^{j-1}K_cDEPEND_m^x$ is c-receive-dependent and initially false. We now claim $K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ is c-receive-dependent. By Lemma 7, introspection, and the Message Chain Theorem, $K_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ is m-receive-dependent. Further, $K_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ is initially false, because $DEPEND_m^x$ is. Therefore, by Lemma 7, introspection, and the Message Chain Theorem, $K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ is c-receive-dependent; further, $K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ is initially false, because $DEPEND_m^x$ is. □

Lemma 16 shows that ① $(K_cK_m)^jK_cDEPEND_m^x$ must hold for arbitrarily deep nesting in order for c to accept, and ② $(K_mK_c)^iDEPEND_m^x$ must hold for arbitrarily deep nesting for m to award to c.

Lemma 16 For any weakly terminating C-system $\mathbf{M}$ which is k-transit bounded and subject to process failures and recovery,

- $\mathbf{M} \models ACCEPT_c \supset (K_cK_m)^jK_cDEPEND_m^x$, for any $c \in \mathcal{C}$, and for all $j \geq 0$.
- $\mathbf{M} \models AWARD_m^c \supset (K_mK_c)^iDEPEND_m^x$, for any $c \in \mathcal{C}$, and for all $i \geq 1$.

Proof: We show the first; the second follows immediately. We prove this by induction on j.

Base case: $j = 0$. The claim that $\mathbf{M} \models ACCEPT_c \supset K_cDEPEND_m^x$ holds by Lemma 9.

Inductive hypothesis: $j > 0$. Assume the inductive hypothesis holds for $j-1$. Therefore, $\mathbf{M} \models ACCEPT_c \supset (K_cK_m)^{j-1}K_cDEPEND_m^x$. We now claim that $\mathbf{M} \models ACCEPT_c \supset K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$. This holds immediately from
(a) an application of the Award Knowledge Rule on $(K_cK_m)^{j-1}K_cDEPEND_m^x$ (which is c-receive-dependent by Lemma 15), and
(b) an application of the Accept Knowledge Rule on the result of application (a) (which is m-receive-dependent, also by Lemma 15). □

Lemma 16 gave knowledge levels required for commitment; Lemma 17 now establishes the accompanying communication requirement of arbitrarily many consecutive message chains.

Lemma 17 In a C-system $\mathbf{M}$ subject to process failures,

- for all $j \geq 0$, if, for some $(e,f) \in \mathbf{Pts}(\mathcal{E})$ and $c \in \mathcal{C}$, $(e,f) \models (K_cK_m)^jK_cDEPEND_m^x$, then there is a sequence of consecutive message chains $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^j$ in $(e,0)$ to (e,f); and
- for all $i \geq 0$, if, for some $(e,f) \in \mathbf{Pts}(\mathcal{E})$ and $c \in \mathcal{C}$, $(e,f) \models (K_mK_c)^iDEPEND_m^x$, then there is a sequence of consecutive message chains $c \xrightarrow{+} m(\xrightarrow{+} c \xrightarrow{+} m)^i$ in $(e,0)$ to (e,f).

Proof: We prove the first by induction on the length of the sequence of chains, j; the second follows analogously by induction on i.

Base case: $j = 0$. The claim is that, for any $(e,f) \in \mathbf{Pts}(\mathcal{E})$, if $(e,f) \models K_cDEPEND_m^x$, then there is a $DEPEND_m^x$-message chain from m to c in interval $(e,0)$ to (e,f). This holds by Lemma 13.

Inductive step: $j > 0$. Assume the inductive hypothesis holds for $j-1$. Now we claim that $K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$ requires a sequence of message chains $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^j$. That is, if $(e,f) \models K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$, then there is a sequence of consecutive message chains $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^j$ in interval $(e,0)$ to (e,f). From the inductive hypothesis, we can assert the existence of the chain sequence $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^{j-1}$, required to establish $(K_cK_m)^{j-1}K_cDEPEND_m^x$. By Lemma 15, $(K_cK_m)^{j-1}K_cDEPEND_m^x$ is c-receive-dependent in **M**. Therefore, by Lemma 7, $(K_cK_m)^{j-1}K_cDEPEND_m^x$ is c-failure-dissociated in **M**. By Lemma 15, $(K_cK_m)^{j-1}K_cDEPEND_m^x$ is initially false. By process introspection, $(K_cK_m)^{j-1}K_cDEPEND_m^x$ is local to c. Therefore, by the Message Chain Theorem, there is a $(K_cK_m)^{j-1}K_cDEPEND_m^x$-message chain $c \xrightarrow{+} m$ in $(e,0)$ to (e,f), to establish $K_m(K_cK_m)^{j-1}K_cDEPEND_m^x$. Now this chain must strictly follow $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^{j-1}$, because $(K_cK_m)^{j-1}K_cDEPEND_m^x$ must hold at the start of the new $c \xrightarrow{+} m$, and $(K_cK_m)^{j-1}K_cDEPEND_m^x$ cannot hold any earlier than the end of $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^{j-1}$. From this, we conclude the existence of $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^{j-1} \xrightarrow{+} m$. By similar reasoning, there is a $K_m(K_cK_m)^{j-1}K_cDEPEND_m^x$-message chain, to establish $K_cK_m(K_cK_m)^{j-1}K_cDEPEND_m^x$. We can conclude the existence of $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^{j-1} \xrightarrow{+} m \xrightarrow{+} c$, or $m \xrightarrow{+} c(\xrightarrow{+} m \xrightarrow{+} c)^j$. □

From the fact that commitment between c and m requires infinitely long sequences of message chains between c and m and the fact that commitment must be possible in any C-system, we conclude our impossibility result.

Theorem 18 There is no weakly terminating C-system which is k-transit bounded and subject to process failures and recovery. □

The analogue of this result for atomic commitment tells us the following: if a round-based atomic commitment protocol is resilient to process failures and recovery and such that a message may be received only in the round in which it is sent, then the protocol may run forever.

From the knowledge levels in Lemma 16, one might think that one can prove Theorem 18 using common knowledge. Indeed, another way to prove this theorem would be to show that commitment in the given systems requires common knowledge among $\{x\}$ and m of $DEPEND_m^x$, which, as a direct corollary of the Message Chain Theorem, is impossible. The current results on attaining common knowledge do not address systems with process failures (although this extension should not be difficult). The Message Chain Theorem allows us to reason about both finite and infinite knowledge levels in several kinds of systems, including systems in which one cannot attain common knowledge. Further, we can reason about incremental gains in a process's knowledge through communication. Finally, the Message Chain Theorem applies to a broad class of problems.

We note that commitment under other system assumptions may require a variant of common knowledge of some proposition; for example, [Hadz88] shows the impossibility of nonblocking atomic commitment under permanent communication failures or weak termination and communication failures by showing ① the need for eventual common knowledge of a particular proposition, and ② the impossibility of attaining that common knowledge in the given systems. Our proof technique using nested knowledge levels and consecutive message chains may also be used directly in this example [Maze89].

7 SUMMARY

Our contribution in this paper is twofold. First, the Message Chain Theorem gives new, general situations in which message chains are required for knowledge gain. This can be used as a tool in examining knowledge and communication requirements for problems in distributed systems, without the need to repeat the lengthy combinatorial proof of the Message Chain Theorem. We note that, with minor changes to the proof, one can show similar message chain results for systems with other behaviours, such as asynchronous communication. Second, we illustrate the use of the Message Chain Theorem in showing impossibility results, by giving a new result for commitment problems, based on a knowledge and communication analysis of the problem specification.

Acknowledgements

Many discussions with Vassos Hadzilacos greatly influenced the work reported here. Questions from Joe Halpern and from Mark Tuttle led to some important clarifications and improvements in presentation. This research has been partially supported by the Natural Sciences and Engineering Research Council of Canada under grant A3356 and the University of Toronto under a Graduate Fellowship.

References

[ChLa85] K.M. Chandy and L. Lamport. "Distributed Snapshots: Determining Global States of Distributed Systems." *ACM Trans. on Computer Systems*, **3**, 1 (February 1985), 63-75.

[ChMi86] K.M. Chandy and J. Misra. "How Processes Learn." *Distributed Computing*, **1**, 1 (1986), 40-52.

[Hadz88] V. Hadzilacos. Class notes, CSC2221F (Topics in Distributed Systems), Department of Computer Science, University of Toronto, Toronto ON, Fall Term 1988.

[Hadz89] V. Hadzilacos. "A Knowledge Theoretic Analysis of Atomic Commitment." Submitted for publication.

[HaMo90] J. Halpern and Y. Moses. "Knowledge and Common Knowledge in a Distributed Environment." To appear in *Journal ACM*. (A preliminary version appears in *Proc. Third ACM Symp. Principles of Distributed Computing*, 1984, 50-61; revised versions appears as Research Report RJ4421, IBM Research Laboratory, San Jose CA, 1986, 1988.)

[KoTo88] R. Koo and S. Toueg. "Effects of Message Loss on the Termination of Distributed Protocols." *Information Processing Letters*, **27**, 4 (1988), 181-88.

[Lamp78] L. Lamport. "Time, Clocks and the Ordering of Events in a Distributed System." *CACM*, **21**, 7 (July 1978), 558-65.

[Maze89] M.S. Mazer. *A Knowledge-Theoretic Account of Negotiated Commitment*, Ph.D. Thesis, Department of Computer Science, University of Toronto, 1989 (available at Technical Report CSRI-237, Computer Systems Research Institute, University of Toronto, 1990).