A Knowledge Theoretic Account Of Recovery In Distributed Systems: The Case Of Negotiated Commitment

Murray S. Mazer

Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 1A4

mazoo@toronto.csnet

ABSTRACT

We are interested in commitment problems in potentially faulty distributed environments; for such problems, the behaviour of failed processes during recovery is relevant to consistency. In particular, we examine *negotiated commitment*, which is the problem of ensuring that each participant in a negotiation reaches a consistent local decision on the outcome. Even undecided, recovering participants must reach a consistent decision on the outcome, because other participants may have committed to an outcome and taken further actions based upon the expected commitment of the recovering participant.

To facilitate the use of knowledge theory to guide the design of protocols for commitment problems, we give an account of process failure and recovery. Using knowledge theory, we show that independent recovery is impossible — i.e., a recovering participant whose decision must be based on some knowledge about other participants in the system cannot decide upon recovering without communicating with other participants. If the participant is in a decided state upon recovery without such communication, then it must have been decided when it failed, and furthermore it must have been decided before it failed. We also give levels of interparticipant knowledge necessary for achieving nonblocking recovery in the absence of total participant failure.

1. INTRODUCTION

Knowledge theory has been used recently to analyse some problems in potentially faulty distributed environments (e.g., [5,7]), but the issue of process *recovery* has been ignored. In *agreement* (or *consensus*) problems, such as Byzantine Agreement, the nonfaulty participants attempt to agree on some value of interest. The post-failure actions or decisions of faulty processes are irrelevant to the consistency of the outcome. Therefore, in designing protocols for such problems, one need not include recovery actions. Dwork and Moses [5], for example, give a knowledge theoretic analysis of Byzantine Agreement protocols in a round-based model of computation which admits crash failures; they discuss a knowledge-based protocol which all correct (nonfailed) processors use to attain Simultaneous Byzantine Agreement. Halpern and Moses [11] and Halpern and Zuck [12] use knowledge theory to analyse the effects of communication failures in distributed environments, but they do not address process failures.

In *commitment* problems, such as Atomic Commitment [6], the behaviour of faulty processes during recovery is *not* ignored; rather, recovering processes are bound by the same consistency requirements as if they had not failed. When designing resilient distributed protocols for problems in which failing processes are *not* excused from consistency requirements, one must actually design three subprotocols: (1) a *failure-free* protocol, to be executed by the participants in executions without failures; (2) a *termination* protocol, to be executed by nonfailed participants which detect failures and are unsure of the outcome of the overall protocol relevant to them, to attempt to terminate consistently; and (3) a *recovery* protocol, to be executed by participants which fail and then recover, to allow such participants to terminate consistently. To facilitate the use of knowledge theory to guide the design of these protocols, we give an account of failure and recovery.

We consider recovery in the context of a problem called *negotiated commitment*, which is the problem of ensuring that all parties in a negotiation commit to a consistent view of the outcome of the negotiation. Negotiated commitment is fundamental to building negotiating systems. In the most basic negotiation (of which other negotiations are elaborations), commitment proceeds as follows: an individual with a task to share (the *manager*) announces a contract embodying the task to its list of potential partners (the *contractors*) and requests bids for accomplishing the task; the contractors reply with bid messages; the manager considers the bids, chooses contractors to receive the contract, and notifies the contractors of its decision. The participants must agree on the outcome of the manager-centric negotiation, to prevent inconsistent (and potentially disastrous) actions being taken based on incorrect perceptions of the outcome. Atomic commitment is a special case of negotiated commitment.

Negotiated commitment arises in supporting dynamic interaction of agents in organizational systems. Examples include automated stock trading systems, distributed planning systems, and distributed transaction systems. Using a knowledge theoretic approach, Mazer [13] addresses issues important to negotiated commitment, such as definitions of consistency, independence of awards, number of awards, binding power of bids, and design of commitment protocols. As our basis for this discussion of recovery, we use one family of negotiated commitment problems in which each contractor is bound by its bid and in which zero or more awards, each independent of the others, may be given in a specific negotiation instance.

Our contributions in this paper include: an introduction to the problem of negotiated commitment; a characterization of process failure and recovery; a knowledge theoretic proof of

the impossibility of independent recovery (while this was known for atomic commitment (e.g., [14]), our result is for a more general problem and uses a different formal tool); and a distinction between nonblocking termination and nonblocking recovery, including a partial characterization of the knowledge required to achieve nonblocking recovery.

Section 2 introduces our event-based model of distributed computations and the constraints which govern the construction of the possible executions of a distributed protocol. Then, in Section 3, we give a temporal knowledge logic to be used in specifying and analyzing negotiated commitment, and we show some key intermediate results on the evolution of knowledge in distributed computations. In particular, we identify an important class of local formulas whose values cannot be known remotely by default (e.g., by failure detection) but only by message receipt. Section 4 presents a formal specification of consistent negotiated commitment; we analyse the specifications to yield knowledge states which must hold when decisions hold. In Section 5, we are finally ready to address recovery. We specify when a process may recover, and we define independent recovery; then we show it to be impossible. This is true because the concomitant knowledge states needed by a recovering process for decision cannot be gained during recovery without communication with other processes. Furthermore, we show that general nonblocking recovery is impossible but that a slightly weaker notion of nonblocking recovery can be achieved. We also characterize levels of knowledge necessary for this kind of nonblocking recovery.

2. MODEL OF DISTRIBUTED SYSTEMS

In this model, adapted from [4,7,8], a distributed system consists of two types of elements: ① *processes*, which can execute events (let Π represent the set of n processes in the system); and ② a *communication system*, $\mathcal{N}$, which contains a set of message packets of the form $\langle p, \underline{m}, q \rangle$, representing a message $\underline{m}$ from process p to process q.*

Each process in a distributed system is characterized by a set of *process executions*, each of which is a sequence of abstract events. These events include: LOCAL (the executing process performs an unspecified internal action with no external communication); SEND($\underline{m}, p$) (the executing process sends message $\underline{m}$ to process p); RECV($\underline{m}, p$) (the executing process receives message $\underline{m}$ from process p); and FAIL (the executing process crash-fails). Given a protocol or algorithm for a process set Π in a distributed environment with specific properties, the distributed system prescribed by that protocol is modeled by the set of possible executions, $\mathcal{E}$, over Π. Each member of $\mathcal{E}$ captures a process execution for each process in Π and the behaviour of the communication system — intuitively, a complete description of an execution of the system [9]. In order to examine the system at various steps in an execution, we introduce a set of *external observation frames*, which allow us to capture each execution as a series of *snapshots*.We identify the set of frames $\mathcal{F}$ with the nonnegative integers.

Formally, an execution $e \in \mathcal{E}$ is a function mapping a frame $f \in \mathcal{F}$ to a snapshot $e(f)$; the set of snapshots of $\mathcal{E}$, denoted $\mathcal{S}(\mathcal{E})$, is $\{e(f) \mid e \in \mathcal{E} \text{ and } f \in \mathcal{F}\}$. Each snapshot maps each process p to a pair, (*history*, *state*), and maps the communication system, $\mathcal{N}$, to a set of message packets (the message packets "in transit" during the snapshot). The *history* is the finite sequence of events executed by p (its partial execution) in e up to frame f; the *state* is

*Assume we have some vocabulary $\underline{\mathcal{M}}$ from which $\underline{m}$ is taken.

the state which p occupies in frame f of execution e. For clarity, we write $e(f,p).history$ and $e(f,p).state$ for the mappings to p's history and state, respectively, in frame f of execution e, and $e(f,\mathcal{N})$ for the mapping to the contents of $\mathcal{N}$. $e(f,p)$ without a "." qualifier means $e(f,p).history$.

We will need some more notation. For $P \subseteq \Pi$, $\overline{P}$ denotes $\Pi - P$. $d \dashv e(f,p)$ means that d is the last event in p's event sequence in snapshot $e(f)$. $e(f+1,p).history = e(f,p).history + d$ indicates the concatenation, in frame $f+1$, of event d onto p's process execution in $e(f)$ (i.e., p executed event d between frames f and $f+1$). For snapshots $t,s \in \mathcal{S}(\mathcal{E})$, $s =_p t$ iff $s(p).history = t(p).history$, and (for $P \subseteq \Pi$) $s =_P t$ iff $s =_p t$ for all $p \in P$. $s =_{\mathcal{N}} t$ iff $s(\mathcal{N}) = t(\mathcal{N})$. $s = t$ iff $s =_\Pi t$ and $s =_{\mathcal{N}} t$. Given $e_1,e \in \mathcal{E}$, and $f \in \mathcal{F}$, $e(f) \equiv e_1(f)$ iff $e(g) = e_1(g)$ for all $0 \leq g \leq f$. For $x \in \Pi \cup \{\mathcal{N}\}$, $e_1(f) \equiv_x e(f)$ is short for $e_1(g) =_x e(g)$ for all $0 \leq g \leq f$. For $X \subseteq \Pi \cup \{\mathcal{N}\}$, $e_1(f) \equiv_X e(f)$ is short for $e_1(g) =_x e(g)$ for all $x \in X$ and $0 \leq g \leq f$. For two snapshots $e(f), e'(g) \in \mathcal{S}(\mathcal{E})$, $e'(g)$ is a *proper extension* (or *extension*) of $e(f)$, denoted $e'(g) > e(f)$ (or $\geq$), if $e'(f) \equiv e(f)$ and $g > f$ (or $\geq$). For $g,f \in \mathcal{F}$ such that $g \geq f$, $e(g,p) - e(f,p)$ yields the suffix of p's history in $e(g)$ obtained by removing p's history in $e(f)$ (that is, the events executed by p between frames f and g).

The formation of valid executions of a distributed system and the snapshots thereof is governed by constraints which reflect certain properties of the system being modelled. We give some here. $e(0,p)$ is the empty sequence Λ for all $p \in \Pi$, and $e(0,\mathcal{N}) = \emptyset$ — the system starts "empty". The sequence of events of any process in one snapshot must be prefixed by, and extend by zero or one events, the sequence of events of that process in the preceding snapshot. Only messages which were sent but not yet received may appear in the message system. A message must be in the message system in the snapshot before that in which it is received. A message packet which has disappeared from the message system may not "magically" reappear. We also assume "honest" messages — i.e., if $\mathsf{SEND}(\phi,p) \dashv s(q)$, then $s \models K_q\phi$. Communication failure is modelled by allowing any message packet in the communication system to disappear (in a similar execution), though no message *must* disappear. Finally, messages from one process to another can only be received in the order sent.

As argued in [6], only *crash failures* make sense in the context of commitment problems. In a crash failure, the failing process stops executing events; if it recovers, it executes events according to its protocol. A FAIL event models a process crash failure. A system $\mathcal{E}$ is *subject to process crash failures* if it satisfies the following system-level constraint:

A Process May (But Need Not) Fail: Given an $e(f) \in \mathcal{S}(\mathcal{E})$ and $p \in \Pi$, if $\mathsf{FAIL} \not\dashv e(f-1,p)$ (i.e., p is not failed in the previous snapshot) and $\Lambda \neq e(f,p) - e(f-1,p) \neq \mathsf{FAIL}$ (i.e., p has just executed some nonfailure event) then there is some $e' \in \mathcal{E}$ such that: ① $e'(f) \equiv_{\Pi\setminus\{p\}} e(f)$; ② $e'(f,p).history = e(f-1,p).history + \mathsf{FAIL}$; ③ $e'(f-1) \equiv_{\mathcal{N}} e(f-1)$; and ④ if $\mathsf{RECV}(\underline{m},q) \dashv e(f,p)$, then $e'(f,\mathcal{N}) = e(f,\mathcal{N}) \cup \{\langle q,\underline{m},p\rangle\}$; otherwise if $\mathsf{SEND}(\underline{m},q) \dashv e(f,p)$, then $e'(f,\mathcal{N}) = e(f,\mathcal{N}) \setminus \{\langle q,\underline{m},p\rangle\}$; otherwise, $e'(f,\mathcal{N}) = e(f,\mathcal{N})$.

If a nonfailed process has just executed some nonfailure event in some execution e in frame f, then there is an execution e' which is identical to e up to frame f except that (1,2) p's last event is replaced by FAIL, and (3,4) if p's last event was RECV (or SEND), then the received message appears in (or does not appear in) the communication system in e'. The following process execution level constraint models executions in which processes cannot recover from

failures: (**No Process Failure Recovery**) For $p \in \Pi$, $f \in \mathcal{F}$, $e \in \mathcal{E}$, if FAIL $\dashv e(f,p)$, then $e(f,p) = e(g,p)$, for all $g > f$ (no event may follow a FAIL event in a process execution).

The ability of a process to execute an event may depend on the events executed by other processes and on the behaviour of the communication system. An important example is the RECV event — a process can execute a RECV event only if there is an appropriate message in the communication system. Fault detection can also affect a process' allowed executions. Consider two snapshots $e(f)$ and $e_1(g)$ such that $e(f) =_P e_1(g)$; it is not necessarily the case that $e(f) =_{\overline{P}} e_1(g)$. We may wish to know if there is an execution $e_2(h)$ such that $e_2(h) =_P e(f+1)$ and $e_2(h) =_{\overline{P}} e_1(g+1)$. We can describe process progress constraints which prescribe such possible fusions following receive, failure-detection, or other events. The constraint needed in this paper, called *Noncommunicative Progress*, says that if none of the processes in P has, in its last event in $e(f+1)$, received a message from, or detected a failure of, a process in $\overline{P}$, then the projections of P from $e(f+1)$ and of $\overline{P}$ from $e_1(g+1)$ may be fused (because the last event of each $p \in P$ does not depend upon the actions of any other elements of the system).

Our model of distributed systems captures a system with *asynchronous* (or *nonblocking*) *sends* and *blocking receives* [1] (cf. [8]). A message, when it is received, contains information about the sender's state that, in general, is not necessarily still its current state. An important exception to this is the passing of messages reflecting *stable* properties — in our analysis of distributed negotiation, we are almost exclusively concerned with stable properties.

3. KNOWLEDGE LOGIC

This temporal modal logic, based on [10], allows us to talk about the knowledge ascribed to processes in a distributed computation and the future states of propositions. The language has the following symbols: a set Φ of primitive propositional variables; a set Π of process names; $\{\sim, \vee, \Diamond, \heartsuit, (,)\}$; $\{K_x \mid x \in \Pi\}$; and $\{K_X \mid X \subseteq \Pi\}$. The set of well-formed formulae $\mathcal{L}_\Pi(\Phi)$ is the smallest set such that (1) every member of Φ is a formula, and (2) if ϕ and ψ are formulae, then so are $(\sim \phi)$, $(\phi \vee \psi)$, $\Diamond\phi$, $\heartsuit\phi$, $K_x\phi$, $K_X\phi$. We abbreviate $(\sim((\sim \phi) \vee (\sim \psi)))$ by $(\phi \wedge \psi)$, and $((\sim \phi) \vee \psi)$ by $(\phi \supset \psi)$. [†] For $X = \{x1, x2, \ldots, xm\}$, and $\psi_{(x)}$ a wff mentioning x, $\widehat{x \in X}\psi_{(x)} \stackrel{\text{def}}{=} \psi_{(x/x1)} \wedge \psi_{(x/x2)} \wedge \ldots \wedge \psi_{(x/xm)}$; that is, the conjunction of instances of ψ with all appearances of x in each instance of ψ replaced uniformly by an element of X.

We use a multiple knower, "possible snapshots" semantics, a possible worlds semantics. A *Kripke model* of the language $\mathcal{L}_\Pi(\Phi)$ is a tuple $\mathrm{M}=(\mathcal{E}, \mathcal{A}, \approx_{p1}, \approx_{p2}, \ldots, \approx_{pn})$, where $\mathcal{E}$ is a system over Π, $\mathcal{A}:\Phi \rightarrow 2^{\mathcal{S}(\mathcal{E})}$, (that is, $\mathcal{A}$ maps each primitive proposition to the set of snapshots in which the proposition holds), and each $\approx_{pi}$ is a binary "snapshot similarity" relation on the snapshots in the system $\mathcal{E}$, one for each process in Π. Given two snapshots $s, t \in \mathcal{S}(\mathcal{E})$, $(s,t) \in \approx_p$ iff $s(p).\text{state} = t(p).\text{state}$. $\approx_{pi}$ divides the set of snapshots into equivalence classes for each p_i. For $P \subseteq \Pi$, we write $s \approx_P t$ iff $s \approx_p t$ for all $p \in P$.

Given a model M, we write $(\mathrm{M}, s) \models \phi$ to express that ϕ holds in snapshot s of the given model. (If M is understood from context, we write $s \models \phi$.) We define $\models$ as follows (assume we are given $e(f) = s \in \mathcal{S}(\mathcal{E})$, and wffs $\phi, \psi \in \mathcal{L}_\Pi(\Phi)$):

1. For $\phi \in \Phi$, $(\mathrm{M}, s) \models \phi$ iff $s \in \mathcal{A}(\phi)$.

[†] In the sequel, we elide the parentheses "(" and ")" in the usual way in formulae in which no ambiguity results.

2. $(\mathbf{M}, s) \models (\sim \phi)$ iff $(\mathbf{M}, s) \models \phi$ does not hold.

3. $(\mathbf{M}, s) \models (\phi \vee \psi)$ iff $(\mathbf{M}, s) \models \phi$ or $(\mathbf{M}, s) \models \psi$ (inclusively).

4. **(Eventually)** $(\mathbf{M}, e(f)) \models \Diamond\phi$ iff, for all $e' \in \mathcal{E}$ such that $e(f) \equiv e'(f)$, there is some $h \geq f$ such that $(\mathbf{M}, e'(h)) \models \phi$ (i.e., iff ϕ is true now or will be in any execution extending s).

5. **(Never)** $(\mathbf{M}, e(f)) \models \heartsuit\phi$ iff, for all $e' \in \mathcal{E}$ such that $e(f) \equiv e'(f)$, $(\mathbf{M}, e'(h)) \models\sim \phi$ for all $h \geq f$ (i.e., ϕ does not hold now and never will in any possible extension of $e(f)$). ‡

6. **(Process Knowledge)** For $p \in \Pi$, $(\mathbf{M}, s) \models K_p\phi$ iff, for all $e'(g) \in \mathcal{S}(\mathcal{E})$ such that $e(f) \approx_p e'(g)$, $(\mathbf{M}, e'(g)) \models \phi$ (i.e., iff ϕ is true in all snapshots which look to p similar to the current one).

7. **(Implicit Knowledge)** For $P \subseteq \Pi$, $(\mathbf{M}, s) \models K_P\phi$ iff for all $e'(g) \in \mathcal{S}(\mathcal{E})$ such that $e(f) \approx_P e'(g)$, $(\mathbf{M}, e'(g)) \models \phi$ (i.e., iff $(\mathbf{M}, e'(g)) \models \phi$ for all $(e(f), e'(g)) \in \approx_{p1} \cap \approx_{p2} \cap \ldots \approx_{pm}$, where $P = \{p1, p2, \ldots, pm\}$).

Notice that $(\mathbf{M}, s) \models K_p\phi$ iff $(\mathbf{M}, s) \models K_{\{p\}}\phi$.

We use a specialized interpretation called a *complete history interpretation* [11] in which $s(p).state = s(p).history$, for all $s \in \mathcal{S}(\mathcal{E})$, $p \in \Pi$. Therefore, $s \approx_p t$ iff $s =_p t$. Recall that $\approx_p$ deals with state similarity and $=_p$ with execution similarity. Under this interpretation, two snapshots s and t from $\mathcal{S}(\mathcal{E})$ look similar to p if p executes the same sequence of events in s and in t. Note that *other* processes may execute different sequences of events in s and in t. In this interpretation, the process' state reflects the most information possible about a process' execution.

Additional Concepts

We present some important additional concepts, based on the computation model and the logic, which we use in our discussion of negotiated commitment and recovery. These results will help us analyse the bidding and decision propositions of the participants.

Locality, Stability, and Nonuniformity

A wff ϕ is *local to* P, for $P \subseteq \Pi$, if, for all $s \in \mathcal{S}(\mathcal{E})$, $s \models K_P\phi \vee K_P \sim \phi$. That is, P always knows the value of ϕ. Local formulae are intended to model predicates whose value is controlled by the actions of the processes to which the formulae are local [4,7]. ϕ is *uniquely local* to $P \subseteq \Pi$ if ϕ is local to P and not local to $\overline{P}$.

The next two results will be important for analysing some of the specifications of negotiated commitment which have the form of the constraint given in each lemma. The first lemma states that if a wff uniquely local to a process q must hold whenever a wff uniquely local to another process p holds, then whenever p's wff holds, p must know that q's wff holds.

‡ "$e(f) \models \heartsuit\phi$" is *not* the same as "$e(f) \models\sim \Diamond\phi$". If we were to define $\Box$ ("Always") in the obvious way parallel to $\heartsuit$, then $\heartsuit\phi$ is the same as $\Box \sim \phi$, but because "never" is a very useful concept in our analysis, we use a single symbol instead of two.

Lemma 1 For all $s \in \mathcal{S}(\mathcal{E})$, $p, q \in \Pi$, ϕ, ψ wffs such that ϕ is uniquely local to p and ψ is uniquely local to q, if the constraint "$s \models \phi \supset \psi$" must hold, then $s \models \phi \supset K_p\psi$.

Proof: By way of contradiction (henceforth, "bwoc"), assume not. Then, for some $s \in \mathcal{S}(\mathcal{E})$, $p, q \in \Pi$, ϕ, ψ as above, $s \models \phi \wedge \sim K_p\psi$. Then there must exist some $t \in \mathcal{S}(\mathcal{E})$ such that $s \approx_p t$ and $t \models \sim \psi$. Since ① ϕ is uniquely local to p, ② $s \models \phi$, and ③ $t \approx_p s$, then $t \models \phi \wedge \sim \psi$, contradicting the given constraint. □

A wff ϕ is *stable* if the following property holds: for all $s \in \mathcal{S}(\mathcal{E})$, if $e(f) \models \phi$, then for all $g > f$, $e(g) \models \phi$. A stable wff stays true forever after it becomes true [11].

Lemma 2 For ϕ, ψ stable wffs, if, for all $e \in \mathcal{E}$, $e(0) \models \Diamond(\phi \wedge \psi)$, then, for all $s \in \mathcal{S}(\mathcal{E})$, $s \models \phi \supset \Diamond\psi$ and $s \models \psi \supset \Diamond\phi$.

Proof: We prove the former; the latter follows analogously. Bwoc, assume $e(0) \models \Diamond(\phi \wedge \psi)$, but $s \models \phi \wedge \sim \Diamond\psi$. Then there is an $e_1 \in \mathcal{E}$ extending s such that $e_1(g) \models \phi \wedge \psi$, for some $g \in \mathcal{F}$, violating the antecedent. □

A wff ϕ is *valid* (or *unsatifiable*) in a system $\mathcal{E}$ if, for all $s \in \mathcal{S}(\mathcal{E})$, $s \models \phi$ (or $s \models \sim \phi$). A wff ϕ is *nonuniform* in a system $\mathcal{E}$ if it is neither valid nor unsatisfiable.

Failure-Detectable Propositions

We give each process $p \in \Pi$ a local predicate $FAILED_p \in \Phi$. For any $s \in \mathcal{S}(\mathcal{E})$, $s \models FAILED_p$ iff $\mathsf{FAIL} \dashv s(p)$. Under our current assumptions, which allow no process recovery, the $FAILED_p$ predicate is stable (this will change when we address process recovery). We use the $\mathsf{PROCDETECT}(p)$ event executed by q to model q's detection of p's failure. Given a snapshot $s \in \mathcal{S}(\mathcal{E})$, $Failed(s) = \{p \mid p \in \Pi \text{ and } \mathsf{FAIL} \dashv s(p)\}$. For $FAILED_p \in \Phi$ and $P \subseteq \Pi$, we abbreviate by $\mathbf{E}_P\phi$ the formula $\widehat{p \in P}(FAILED_p \vee K_p\phi)$; that is, every process in P is failed or knows ϕ [7].

A wff ϕ, uniquely local to $p \in \Pi$, is *failure-detectable* by $q \in \Pi$ ($q \neq p$) if, for all $e(f) \in \mathcal{S}(\mathcal{E})$, if ① $e(f-1) \models \sim K_q\phi$ and $\mathsf{PROCDETECT}(p) \not\dashv e(f-1, q)$ and ② $\mathsf{PROCDETECT}(p) \dashv e(f, q)$, then $s \models K_q\phi$. That is, the act of detecting p's failure leads q to know ϕ. For example, $FAILED_p$ is failure-detectable. A wff ϕ uniquely local to p is called *failure-insecure* if the following property holds: for all $s \in \mathcal{S}(\mathcal{E})$, if $\mathsf{FAIL} \dashv s(p)$, then $s \models \sim \phi$. A wff ϕ uniquely local to p is called *failure-ensured* if the following property holds: for all $s \in \mathcal{S}(\mathcal{E})$, if $\mathsf{FAIL} \dashv s(p)$, then $s \models \phi$. A wff ϕ uniquely local to p is *failure-unrelated* if ϕ is neither failure-ensured nor failure-insecure. The following lemmas show that failure of a process p cannot determine the values of any two distinct wffs which are stable, nonuniform, uniquely local to p, and cannot hold simultaneously. Negotiated commitment involves bidding and decision propositions of exactly this type.

Lemma 3 Given $p \in \Pi$, wffs ϕ, ψ both stable, nonuniform, and uniquely local to p, if, for all $e \in \mathcal{E}$, $e(0) \models \Diamond(\phi \wedge \psi)$, then ① neither ϕ nor ψ is failure-ensured, and ② neither ϕ nor ψ is failure-insecure. □

Lemma 4 If a nonuniform wff ϕ uniquely local to p is failure-unrelated, then ϕ is non-failure-detectable. □

Therefore, "For all $e \in \mathcal{E}$, $e(0) \models \Diamond(\phi \wedge \psi)$", where ϕ and ψ are stable, nonuniform, and uniquely local to the same process, tells us that ϕ and ψ are failure-unrelated and non-failure-detectable.

Nondefault Propositions

In some systems, processes may get to know the value of certain propositions local to other processes without receiving any messages (e.g., by detecting failures) [7]. Nondefault wffs are facts for which this is not possible. Several important propositions in the specification of negotiated commitment are provably nondefault. A wff ϕ is called *nondefault* if the following property holds: for all $s \in \mathcal{S}(\mathcal{E})$, $P \subset \Pi$ such that ϕ is uniquely local to $\overline{P}$, if $e(f) \models\sim (K_P\phi \vee K_P \sim \phi)$ and, for all $p \in P$, $\overline{p} \in \overline{P}$, $e(f+1,p) - e(f,p) \neq \mathsf{RECV}(\underline{m}, \overline{p})$, for all $\underline{m}$, then $e(f+1) \models\sim (K_P\phi \vee K_P \sim \phi)$. That is, a process set P cannot come to know a nondefault wff uniquely local to $\overline{P}$ without receiving a message.

Theorem 5 If ϕ is uniquely local to q and is non-failure-detectable, then ϕ is nondefault.

Proof: Assume not. (If ϕ is uniform, then nondefaultness is trivial; assume nonuniform.) Then choose $e(f) \in \mathcal{S}(\mathcal{E})$ such that, for some $p \in \Pi\backslash\{q\}$, $e(f-1) \models\sim (K_p\phi \vee K_p \sim \phi)$ and $e(f,p) - e(f-1,p) \neq \mathsf{RECV}(\underline{m}, \overline{p})$ for all $\overline{p} \in \Pi \setminus \{p\}$, and $e(f) \models K_p \sim \phi$. Therefore, p does not receive a message, yet it comes to know $\sim \phi$. (Choose $K_p \sim \phi$ without loss of generality; the rest of the proof follows if one switches $K_p\phi$ for $K_p \sim \phi$ and ϕ with $\sim \phi$.)

Since $e(f-1) \models\sim K_p \sim \phi$, there is $e_1(g) \in \mathcal{S}(\mathcal{E})$ such that $e_1(g) \approx_p e(f-1)$ and $e_1(g) \models\sim\sim \phi$, or $e_1(g) \models \phi$. Choose $e_2(h) \in \mathcal{S}(\mathcal{E})$ such that $e(f) \approx_p e_2(h)$ and $e_1(g) \approx_q e_2(h)$ (possible because of noncommunicative progress and process execution prefix extension).

Now $e_2(h) \models K_p(\sim \phi) \wedge \phi$, which is impossible. [§] □

Notice that if $FAILED_q$ is not (assumed to be) stable and failure-detectable, then $K_pFAILED_q$ may never hold; p must always be unsure of whether q is live or failed [4]. Stability is needed because authentic failure detection requires that $FAILED_q$ still hold at the snapshot of detection, and failure detection may take arbitrarily long to occur after the failure. Failure-detectability is required to achieve the knowledge level $K_pFAILED_q$, for $p \neq q$.

4. NEGOTIATED COMMITMENT

Recall the simple negotiation described in Section 1. For concreteness, we say that each contractor in a distributed negotiation may choose immutably only one of two bidding options (based on the contract announcement): *Bid* or *No-Bid*. Each contractor can reach exactly one of two immutable *decisions* on the negotiation: *Accept* or *Refuse*[¶]. The manager reaches one immutable *decision* for each contractor: either *Award* or *Reject*. The decision of a contractor c is *consistent* with the manager's decision for c if m decides Award (resp., Reject) for c and c decides Accept (resp., Refuse). The decisions are *inconsistent* if m decides Award (resp., Reject) for c and c decides Refuse (resp., Accept). The decisions of two contractors are implicitly mutually consistent, by definition.[‖]

[§] This theorem can be used to reprove the knowledge gain and loss theorems of [4,9] for non-failure-related process knowledge. Furthermore, if we extend the notion of potential causality from [4,9] to include process failure, then the knowledge gain theorem still holds, under the assumption of **No Process Failure Recovery**. The knowledge loss theorem is unchanged by the addition of failure, because failure-detectable propositions must be stable for failure-detectability to hold.

[¶] Loosely speaking, *Accept* means that the contractor commits to continuing on to perform the contract. *Refuse* means that the contractor commits to not performing the contract.

[‖] This is true for independent awards. The definition of intercontractor consistency changes for negotiated commitment with dependent awards (such as atomic commitment) — see [13].

In our description of negotiating systems (*N-systems*), we have a set $\mathcal{C}$ of contractor processes and a singleton set $\mathcal{M}$ (=$\{m\}$) for the manager process. $\mathcal{C} \cap \mathcal{M} = \emptyset$, and $\Pi = \mathcal{M} \cup \mathcal{C}$. For each $c \in \mathcal{C}$, we define four primitive propositions, BID_c, $NO\text{-}BID_c$, $ACCEPT_c$, and $REFUSE_c$, each of which is stable and uniquely local to c. For the manager process m, we define two primitive propositions for each contractor c, $AWARD_m^c$and $REJECT_m^c$, each of which is stable and uniquely local to m. The stability of the propositions reflects the immutability of the decisions they represent (that is, the commitment). We begin our consideration of the negotiation after the initial contract announcements have been sent out and received by the members of $\mathcal{C}$. Each contractor will eventually set its bid or fail. We want the bids to be set independently of each other, after the system begins execution — this reflects the lack of collusion assumed. If a process is able to bid (or to not bid) in some snapshot of an execution, then it is able to not bid (or to bid) in a snapshot of another execution which, up to the bidding snapshot, appears the same as the first one to all other processes and to the communication system. Our bid constraints, in combination with the **A Process Need Not Fail** constraint, yield that all combinations of bids are possible.

Now we give some of the conditions in the specification of negotiated commitment; we require these for our later results.

Failure-free Decisions: For all $c \in \mathcal{C}$, there is ① an $e \in \mathcal{E}$ such that e contains no process failures or communication failures and (for some $f \in \mathcal{F}$) $e(f) \models AWARD_m^c$and (for some $g \in \mathcal{F}$) $e(g) \models (ACCEPT_c \vee REFUSE_c)$; and ② an $e_1 \in \mathcal{E}$ such that e_1 contains no process failures or communication failures and (for some $h \in \mathcal{F}$) $e_1(h) \models REJECT_m^c$ and (for some $i \in \mathcal{F}$) $e_1(i) \models (ACCEPT_c \vee REFUSE_c)$.
(For each contractor, there is some (at least one) execution without failures in which both the manager and the contractor will decide.)

Post-Failure Decisions: For all $e \in \mathcal{E}$ and $f \in \mathcal{F}$, if $Failed(e(f)) = \emptyset$, then there are $e_1 \in \mathcal{E}$ and $h \in \mathcal{F}$ such that $e_1(f) \equiv e(f)$, there are no process or communication failures in $e_1(i)$ for $f \leq i \leq h$, and $e_1(h) \models \widehat{c \in \mathcal{C}}(ACCEPT_c \vee REFUSE_c) \wedge \widehat{c \in \mathcal{C}}(AWARD_m^c \vee REJECT_m^c)$.
(If no process is now failed and if no new process or communication failures occur for sufficiently long, then all processes will decide.)

No Unilateral Awards: For all $c \in \mathcal{C}$ and $s \in \mathcal{S}(\mathcal{E})$, $s \models AWARD_m^c \supset BID_c$.
(The manager can award a contract only to a bidding contractor.)

Nontrivial Award Decision: For all $e(f) \in \mathcal{S}(\mathcal{E})$ such that $e(f-1) \models\sim K_m BID_c$ and $e(f) \models K_m BID_c$, there is $e_1(g) > e(f)$ such that $e_1(g) \models REJECT_m^c$.
(Any time the manager gets a bid from a contractor c, there is an extension in which m can reject c.)

Decision Harmony proscribes inconsistent commitment decisions.

Decision Harmony: For all $c \in \mathcal{C}$ and $e \in \mathcal{E}$,
① $e(0) \models \Diamond(AWARD_m^c \wedge REFUSE_c)$; ② $e(0) \models \Diamond(REJECT_m^c \wedge ACCEPT_c)$
(① and ② insist that m and c do not decide inconsistently; e.g., ① states that in no execution may m award to c and c refuse.)
③ $e(0) \models \Diamond(AWARD_m^c \wedge REJECT_m^c)$; ④ $e(0) \models \Diamond(ACCEPT_c \wedge REFUSE_c)$
(only one of two possible decisions is allowed).

An N-system $\mathcal{E}$ is called *nonblocking* if, for all $s \in \mathcal{S}(\mathcal{E})$,
$s \models \widehat{c \in \mathcal{C}} \Diamond (FAILED_c \vee ACCEPT_c \vee REFUSE_c) \wedge \widehat{c \in \mathcal{C}} \Diamond (FAILED_m \vee AWARD_m^c \vee REJECT_m^c)$.
That is, each process eventually fails or decides. Informally, a process is *blocked* when it must await the repair of failures before proceeding [14,2]. Blocking is undesirable, because it may cause participants to wait for an arbitrarily long time before deciding consistently, making a contract undecided for arbitrarily long at the blocked participant's site, uselessly holding resources. (As shown in [13] for negotiated commitment, and as known for other problems as well [11], failure-free communications are required to achieve nonblocking.)

We can analyse the above specifications to yield insights into concomitant knowledge states. For example, by Lemmas 3 and 4, Theorem 5, and **Decision Harmony**, we can conclude that the decision propositions are nondefault. Similarly, BID_c and $NO\text{-}BID_c$ are nondefault. Further, $\heartsuit\phi$, for $\phi \in \{BID_c, NO\text{-}BID_c, AWARD_m^c, REJECT_m^c, ACCEPT_c, REFUSE_c\}$, is nondefault. By Lemma 1 and **No Unilateral Awards**, we can conclude that, for any N-system $\mathcal{E}$, any $s \in \mathcal{S}(\mathcal{E})$, and any $c \in \mathcal{C}$, $s \models AWARD_m^c \supset K_m BID_c$. From these results, we can conclude that, before m can award to c, m must receive a message giving c's bid. This is beginning to prescribe some of the message exchange required in a protocol for negotiated commitment.

The following result shows the most general state of knowledge which must hold locally if a process decides without risk of inconsistency.

Theorem 6 For any N-system $\mathcal{E}$, for all $s \in \mathcal{S}(\mathcal{E})$, $c \in \mathcal{C}$, ① $s \models REFUSE_c \supset K_c \heartsuit AWARD_m^c$; ② $s \models REJECT_m^c \supset K_m \heartsuit ACCEPT_c$; ③ $s \models ACCEPT_c \supset K_c \heartsuit REJECT_m^c$; ④ $s \models AWARD_m^c \supset K_m \heartsuit REFUSE_c$.

Proof: We will prove this for the first of the four claims — the proofs for the other three are analogous. Assume bwoc that such a system exists. Then $s \models REFUSE_c \wedge (\sim K_c \heartsuit AWARD_m^c)$; therefore, there is $e'(g) \in \mathcal{S}(\mathcal{E})$ such that $e'(g) \approx_c e(f)$ and $e'(g) \models \sim \heartsuit AWARD_m^c$, so there is an extension of $e'(g)$, say $e''(h)$, such that $e''(h) \models AWARD_m^c$. By similarity and stability, $e''(h) \models REFUSE_c$. This violates **Decision Harmony**. □

We can also show that the above implications cannot be equivalences in some contexts, such as nonblocking [13]. (If they were equivalences, then, as soon as a process reached the consequent level of knowledge, the antecedent proposition would hold; but, for example, the consequent knowledge level is not sufficient to achieve nonblocking.)

Assume that we can identify a level of knowledge (that is, a wff involving knowledge) for a process which is both necessary and sufficient for that process to decide without risk of inconsistency. We assume that all processes will decide (the decision proposition will hold) in the same snapshot in which the identified knowledge holds (this means that the process executes no superfluous events before deciding — cf. the nondominated atomic commitment protocols of [8]).

5. RECOVERY

We want to allow a process which has failed to recover eventually. We allow recovery only when the system has reached some stable or equilibrium point **. The equilibrium point we choose is

**This is essential for reasoning about termination, during which we assume that $FAILED_p$ is stable. While this assumption is not strictly valid (otherwise, we could not have recovery!), stability of $FAILED_p$ allows detecting

that in which all nonfailed participants have decided as much as possible given the current set of failed processes — that is, all nonfailed processes which have not decided will never decide in any extension in which the currently failed processes are still failed [2].††

For $e(f) \in \mathcal{S}(\mathcal{E})$, let *CDecided(e(f))*=$\{c \mid c \in \mathcal{C}$ and $e(f) \models ACCEPT_c \vee REFUSE_c\}$, and let *MDecided(e(f))*=$\{c \mid c \in \mathcal{C}$ and $e(f) \models AWARD_m^c \vee REJECT_m^c\}$. We say p *may recover in snapshot* $e(f)$ if ① $p \in$ *Failed*$(e(f))$, and ② for all $e'(g) > e(f)$ such that *Failed*$(e'(h)) \cap$ *Failed*$(e(f))$ = *Failed*$(e(f))$, $f \leq h \leq g$: *CDecided(s)* = *CDecided(e'(g))* and *MDecided(s)* = *MDecided(e'(g))*. That is, p can recover in snapshot $e(f)$ if, in all consecutive snapshots extending $e(f)$ such that (at least) the currently failed processes are still failed, the currently undecided live processes are still undecided (no process has further decided).

To discuss process failure recovery, we must loosen our **No Process Failure Recovery** constraint. A process with a nonFAIL event following FAIL in its process execution is no longer failed. To model recovery, we say that a RECOVER event is executed (we use a distinct recovery event to make explicit the action taken). Therefore, we say that p is in its *initial recovery stage* in snapshot $e(f)$ if RECOVER $\dashv e(f,p)$. The following conditions apply to recovery: ① (**Authentic Recovery**) if $e(f,p) - e(f-1,p)$ = RECOVER, then p may recover in snapshot $e(f-1)$; and if FAIL $\dashv e(f-1,p)$ and FAIL $\not\dashv e(f,p)$, then RECOVER $\dashv e(f,p)$. ② (**Nontrivial Recovery**) If p could recover in snapshot $e(f-1)$, then there is $e_1 \in \mathcal{E}$ such that $e(f-1) \equiv e_1(f-1)$ and RECOVER $\dashv e_1(f,p)$; also, there is $e_2 \in \mathcal{E}$ such that $e(f-1) \equiv e_2(f-1)$ and FAIL $\dashv e_2(f,p)$ (p may recover, but it need not).

Independent Recovery

Independent recovery is the ability of a process to decide harmoniously after failure without sending or receiving any messages. For $c \in \mathcal{C}$, c *can recover independently in snapshot* s if c is in its initial recovery stage in s and $s \models ACCEPT_c \vee REFUSE_c$. That is, c must have decided one way or the other. The manager m can recover independently in snapshot s if m is in its initial recovery stage at snapshot s and, for all $c \in \mathcal{C}$, $s \models (AWARD_m^c \vee REJECT_m^c)$. An N-system $\mathcal{E}$ supports independent recovery if, for all $p \in \Pi$ and all $s \in \mathcal{S}(\mathcal{E})$, p can recover independently in s.

Theorem 7 There is no N-system supporting independent recovery.

Proof: (We show this for $c \in \mathcal{C}$.) Find $e(f) \in \mathcal{S}(\mathcal{E})$ such that $e(f) \models ACCEPT_c \wedge AWARD_m^c$. Therefore, at least $e(f) \models K_c \heartsuit REJECT_m^c \wedge K_m BID_c$. Now find $e(g) < e(f)$ such that $e(g-1) \models\sim K_m BID_c$ and $e(g) \models K_m BID_c$. Therefore, $e(g) \models\sim K_c \heartsuit REJECT_m^c \wedge \sim K_c \heartsuit AWARD_m^c$ (because of **Nontrivial Award Decisions**).

processes to infer certain system knowledge states necessary for decision. A stability assumption has also been necessary for problems such as deadlock detection and computation restart [3].

††In [14], Skeen talks of a more complicated recovery strategy in which a process may recover at any time and attempt to rejoin operational sites executing the termination protocol. If a centralized termination protocol is used, then the recovering process must find the current coordinator and send the prescribed message indicating its local state. The coordinator responds with a new state for the recovering process to occupy. Until the coordinator responds, the process has not fully recovered and cannot be considered an active partner in the protocol. The conclusion Skeen draws (p. 135) is that there is little to be gained by allowing processes to rejoin the termination protocol. Our equilibrium assumption above corresponds to the situation in which no coordinator responds until all known live participants in the (termination) protocol have terminated if possible.

Therefore, there is $e_1(h) \in \mathcal{S}(\mathcal{E})$ such that $e_1(h) \approx_c e(g)$ and $e_1(h) \models \sim \Diamond REJECT_m^c$, so there is $e_2(i) \in \mathcal{S}(\mathcal{E})$ such that $e_2(i) \geq e_1(h)$ and $e_2(i) \models REJECT_m^c$. Similarly, there is $e_3(j) \in \mathcal{S}(\mathcal{E})$ such that $e_3(j) \approx_c e(g)$ and $e_3(j) \models \sim \Diamond AWARD_m^c$, so there is $e_4(k) \in \mathcal{S}(\mathcal{E})$ such that $e_4(k) \geq e_3(j)$ and $e_4(k) \models AWARD_m^c$.

Now let e_5 be such that $e_5(g) \equiv e(g)$ and $e_5(g+1, c) = e_5(g, c) + \mathsf{FAIL}$. Pick $l > g$ such that $e_5(l)$ is c's initial recovery stage. Now $e_5(g) \approx_c e(g)$, so $e_5(g) \models \sim K_c \Diamond REJECT_m^c \wedge \sim K_c \Diamond AWARD_m^c$. Similarly, by nondefaultness of $REJECT_m^c$ and of $AWARD_m^c$, $e_5(l) \models \sim K_c \Diamond REJECT_m^c \wedge \sim K_c \Diamond AWARD_m^c$ (i.e., neither FAIL nor RECOVER could have yielded $K_c \Diamond REJECT_m^c$ or $K_c \Diamond AWARD_m^c$.)

Therefore, $e_5(l) \models \sim ACCEPT_c \wedge \sim REFUSE_c$, or $e_5(l) \models \sim (ACCEPT_c \vee REFUSE_c)$. □

To understand why this result holds, recall that we, the external observers of a distributed system, ascribe knowledge to all processes, including a process whose last event in the snapshot we are examining is FAIL. Furthermore, the decision propositions, such as $ACCEPT_c$ or $REFUSE_c$, are ascribed by us to the process based on the knowledge ascribed to process c in each snapshot. We know that ϕ and $\Diamond\phi$, for $\phi \in \{AWARD_m^c, REJECT_m^c, ACCEPT_c, REFUSE_c\}$ are both nondefault. Therefore, for a process to decide, the process must receive a message telling it (at least) the most general opposite knowledge level we gave in Theorem 6, or something from which that can be inferred.[‡‡] Therefore, if the last event for c in a snapshot is FAIL, then, in terms of the propositions of interest for decision, c will not gain any more knowledge from the FAIL event than it had from its previous event, nor will it gain any more knowledge in its initial recovery stage (i.e., when c doesn't receive or send anything, just shakes off the cobwebs and looks around).

In other words, if $ACCEPT_c$ or $REFUSE_c$ or other nondefault propositions are going to be ascribed to c in the FAIL snapshot, then the same propositions must hold for c in at least the event before the FAIL — FAIL cannot yield the level of knowledge about the local state of another process needed to decide consistently. The same argument holds for RECOVER. Therefore, independent recovery is not possible. If a process has decided upon recovery, then it must have decided before failing. That is, if $\mathsf{FAIL} \dashv e(f, c)$ and $e(f) \models ACCEPT_c$, then $e(f-1) \models ACCEPT_c$ (and similarly for propositions $REFUSE_c$, $AWARD_m^c$, and $REJECT_m^c$, and for $K_m BID_c$ and $K_m NO\text{-}BID_c$). If $\mathsf{RECOVER} \dashv e(f, c)$ and $e(f) \models ACCEPT_c$, then $e(f-1) \models ACCEPT_c$ (and similarly for $REFUSE_c$, $AWARD_m^c$, and $REJECT_m^c$, and for $K_m BID_c$ and $K_m NO\text{-}BID_c$).

When designing recovery protocols, we take into account the level of knowledge which holds for the recovering process when it executes its RECOVER event — for the knowledge relevant to deciding, that turns out to be the knowledge which holds in the snapshot before the FAIL event from which the process is recovering [13].

Nonblocking Recovery

Given that we cannot have independent recovery in an N-system, we ask whether we can have nonblocking recovery. That is, can a recovering process always decide, assuming it does not fail,

[‡‡] In a reduced-view interpretation, in which $s(p).state \neq s(p).history$, executing the RECV event is not enough to ensure that the state of local knowledge inferred from the received message holds — the process must actually be in a state which explicitly reflects having received the message, even if the process then fails (for example, receiving and then writing the message to stable storage — only after the write is completed is the process in the desired state).

no matter what other processes do? The answer is no. One of the scenarios under which we cannot have nonblocking recovery is total failure. Given an N-system $\mathcal{E}$, a snapshot $s \in \mathcal{S}(\mathcal{E})$ *features total failures* if $Failed(s) = \Pi$. An N-system $\mathcal{E}$ *features total failures* if any $s \in \mathcal{S}(\mathcal{E})$ features total failures.

Theorem 8 There is no nonblocking N-system featuring total failures.

Proof: Assume bwoc that such a system exists. Find a failure-free $e(f)$ such that $e(f) \models ACCEPT_c \wedge AWARD_m^c$. Therefore, $e(f) \models K_m BID_c$. Now find $g < f$ such that $e(g-1) \models\sim K_m BID_c$ and $e(g) \models K_m BID_c$. Therefore, $e(g) \models\sim K_c \Diamond REJECT_m^c \wedge \sim K_c \Diamond AWARD_m^c$.

Now find $e_2 \in \mathcal{E}$ such that $e_2(g) \equiv e(g)$ and $e_2(g+1, p) = e(g, p) + \mathsf{FAIL}$ for all $p \in \Pi$ [total failure occurs]. Now find $e_3(h) > e_2(g)$ such that $e_3(h, c) - e_3(h-1, c) = \mathsf{RECOVER}$ and $\mathsf{RECOVER} \notin e_3(h, p) - e_3(g, p)$ for all $p \in \Pi \backslash \{c\}$ (i.e., c is the first to recover).

Proceeding as in the proof of theorem 7, $e_3(h) \models\sim K_c \Diamond REJECT_m^c \wedge \sim K_c AWARD_m^c$. We know that c cannot recover independently; c must communicate with others. By the choice of the execution e_3, c will not receive any further messages about m's knowledge of c's bid unless some other process sends one. Without loss of generality, we may assume e_3 extends $e_3(h)$ such that $\mathsf{RECOVER} \not\in e_3(i, p)$ for all $p \in \Pi$, all $i \geq h$. Therefore, $e_3(i) \models\sim (REFUSE_c \vee ACCEPT_c)$, for all $i \geq h$. □

Therefore, c is blocked at least until it can receive some messages from other processes. For a recovery protocol to be nonblocking, *at least one process must be correct* (i.e., not yet failed) to aid the recovering one(s). Skeen [14] alludes to this in his discussion of nonblocking recovery strategies for atomic commitment.

Even if we do not have total failure, we may not yet have nonblocking recovery. *Weak* nonblocking recovery is nonblocking recovery in the absence of total failures; we assume weak nonblocking recovery in the remaining discussion. We must distinguish between nonblocking behaviour under a no-recovery assumption and nonblocking in a system which admits recovery. In the former, a process p which fails satisfies the nonblocking requirement. Any process q which must be explicitly consistent with p can continue, using a termination protocol, to decide or to fail, thereby satisfying nonblocking. Because p will never recover, it will never need to know anything about q's behaviour while p was failed. If we allow p to recover, however, p will need to know about q's behaviour while p was failed, to ensure that p does not decide inconsistently. If q fails before p can communicate with q, then p must communicate with others about q's actions. If q does not tell others of its decision before q fails, then no process will be able to help p, so p must block. Therefore, q must ensure, before it decides, that every process will eventually know what q's decision direction is or will fail.

Theorem 9 If an N-system $\mathcal{E}$ is nonblocking for recovery at snapshot $s \in \mathcal{S}(\mathcal{E})$, then

1. $\cup_{g<f} Failed(e(g)) \neq \Pi$ (some process(es) did not yet fail).
2. if $s \models AWARD_m^c \vee REJECT_m^c$, then $s \models K_m \Diamond \mathbf{E}_\Pi(\Diamond REJECT_m^c \vee \Diamond AWARD_m^c)$.
   (The manager, when decided about c, must know that eventually all processes will know a direction for c or fail.)
   Proof: Assume bwoc that $s \models AWARD_m^c \vee REJECT_m^c$, but $s \models\sim K_m \Diamond \mathbf{E}_\Pi(\Diamond REJECT_m^c \vee \Diamond AWARD_m^c)$. Then there is $e_1(g) \in \mathcal{S}(\mathcal{E})$ such that $e_1(g) \approx_m s$ and $e_1(g) \models\sim \Diamond \mathbf{E}_\Pi(\Diamond REJECT_m^c \vee \Diamond AWARD_m^c)$. Therefore, there is $e_2 \in \mathcal{E}$ such that $e_2(g) \equiv e_1(g)$ and, for all $h \geq g$, $e_2(h) \models\sim \mathbf{E}_\Pi(\Diamond REJECT_m^c \vee \Diamond AWARD_m^c)$. That is, $e_2(h) \models\sim [\widehat{p \in \Pi}(FAILED_p \vee K_p[\Diamond REJECT_m^c \vee$

$\heartsuit AWARD_m^c]$)]. Therefore, there is $p \in \Pi$ such that $e_2(h) \models \sim (FAILED_p \vee K_p[\heartsuit REJECT_m^c \vee \heartsuit AWARD_m^c])$. Therefore, p does not know the directions for c and has not failed. Now assume that c is recovering in $e_2(h)$ and $e_2(h) \models\sim K_c(\heartsuit REJECT_m^c \vee \heartsuit AWARD_m^c)$. c needs direction from another process; p cannot direct c to decide. Assume without loss of generality that FAIL $\dashv e_2(h, q)$ for all $q \in \Pi \setminus \{p, c\}$. Then no process can direct c to decide. Therefore, c is blocked.

3. if $s \models ACCEPT_c \vee REFUSE_c$, then $s \models K_c \Diamond \mathbf{E}_\Pi(\heartsuit REFUSE_c \vee \heartsuit ACCEPT_c)$.
(A decided contractor must know that eventually all processes will know a direction for c or fail.)
Proof. Analogous to the proof of item two. □

These necessary knowledge levels for nonblocking recovery are stronger than the levels for nonblocking termination (i.e., nonblocking under the **No Process Failure Recovery** assumption). This is essentially because, for termination, the deciding process p need only know that any process with which it must be explicitly consistent will eventually know p's direction or fail. Here, we require that p knows that *all* processes will eventually know p's direction or fail. This knowledge level is not, however, sufficient to achieve nonblocking recovery.

Dependent negotiated commitment requires that, along with the specification of manager-contractor decision harmony given above, each member of manager-chosen *contractor dependency sets* (nonintersecting subsets of $\mathcal{C}$) must be harmonious with each other. The levels of knowledge given in Theorem 9 can be generalized for dependent negotiated commitment by requiring that eventually all participants will either know a direction for *all codependents in c's dependency set* or fail. (In the independent scenario, each dependency set is a singleton.) In atomic commitment, the contractor codependency set is $\mathcal{C}$; nonblocking termination protocols are such that any deciding process knows that eventually all processes will know the decision direction for all processes (or fail) and can therefore tell an uncertain process its direction. Therefore, nonblocking atomic commitment protocols achieve the levels of knowledge identified above for nonblocking recovery. This is likely the reason that the distinction between no-recovery nonblocking termination and nonblocking recovery has not been addressed previously.

6. SUMMARY

We discussed process recovery in faulty distributed environments, in the context of the problem of negotiated commitment. We presented a computation model and a temporal knowledge logic in which we specified negotiated commitment and found some clues, in the form of required knowledge states, to the message passing required in a negotiated commitment protocol. We defined process failure, recovery, and independent recovery. Using knowledge theory, we showed that independent recovery is impossible in general, and we gave knowledge levels necessary for weak nonblocking recovery. The proofs of the concomitant knowledge states and of the impossibility of independent recovery and general nonblocking recovery depended upon nondefaultness, an important property of certain propositions.

Acknowledgements

This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A3356. The author thanks Gerhard Lakemeyer, Vassos Hadzilacos, and Fred Lochovsky for careful readings of earlier versions, and Hector Levesque for comments on an earlier paper which motivated some of the results reported here.

References

[1] G.R. Andrews and F.B. Schneider. "Concepts and Notations for Concurrent Programming." *ACM Computing Surveys*, **15**, 1 (March 1983), 3-43.

[2] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*, Reading MA: Addison-Wesley Publishing Company, 1987.

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

[4] K.M. Chandy and J. Misra. "How Processes Learn." *Distributed Computing*, **1**, 1 (1986), 40-52. (A preliminary version appears in *Proc. Fourth ACM Symp. on Principles of Distributed Computing*, August 1985, 204-14.)

[5] C. Dwork and Y. Moses. "Knowledge and Common Knowledge in a Byzantine Environment I: Crash Failures." *Proc. Conf. on Theoretical Aspects of Reasoning About Knowledge*, Asilomar CA, March 1986, 149-69.

[6] V. Hadzilacos. "On the Relationship Between the Atomic Commitment Problem and Consensus Problems." *Proc. Workshop on Fault-Tolerant Distributed Computing*, March 1986, Asilomar CA. (to be published by Springer-Verlag.)

[7] V. Hadzilacos. "A Knowledge Theoretic Analysis of Atomic Commitment Protocols (Preliminary Report)." *Proc. ACM Symp. Principles of Database Systems*, 1987.

[8] V. Hadzilacos. "A Knowledge Theoretic Analysis of Atomic Commitment." Submitted for publication, 1987.

[9] J.Y. Halpern. "Using Reasoning About Knowledge to Analyze Distributed Systems." in *Annual Review of Computer Science, Vol. 2*, Ed. J.F. Traub, Annual Reviews, Inc., 1987. (also appeared as Research Report RJ5522, IBM Research Laboratory, Almaden CA.)

[10] J. Halpern and Y. Moses. "A Guide to the Modal Logics of Knowledge and Belief: A Preliminary Draft." *Proc. Int'l Joint Conf. on Artificial Intelligence*, 18-23 August 1985, Los Angeles CA, 480-90.

[11] 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; a revised version appears as Research Report RJ4421, IBM Research Laboratory, San Jose CA, 1986.)

[12] J.Y. Halpern and L. Zuck. *A Little Knowledge Goes A Long Way: Simple Knowledge-based Derivations and Correctness Proofs for a Family of Protocols*, Research Report RJ5857, IBM Research Laboratory, Almaden CA, 1987.

[13] M.S. Mazer. Ph.D. Thesis, Department of Computer Science, University of Toronto, 1987 (in progress).

[14] M.D. Skeen. *Crash Recovery in a Distributed Database System*. Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley CA, 1982.