RJ 7727 (71592) 10/3/90
Computer Science

R 42

Research Report

# A UNIFIED THEORY OF DISTRIBUTED COORDINATION
WITH COMMUNICATION UNCERTAINTY

Ray Strong

Danny Dolev

Flaviu Cristian

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

RJ 7727 (71592) 10/3/90
Computer Science

# A UNIFIED THEORY OF DISTRIBUTED COORDINATION

WITH COMMUNICATION UNCERTAINTY

Ray Strong

Danny Dolev

Flaviu Cristian

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

**ABSTRACT:** A prominent piece of distributed systems folklore holds that safety and timeliness are incompatible properties for distributed coordination protocols. In this paper we provide a framework for understanding and comparing various lower bounds and impossibility results concerning the solution of coordination problems like atomic commit for distributed transactions and other consensus problems. Our treatment provides an abstract notion of information transfer that applies equally to systems with message based communication and systems with other information transfer mechanisms, such as shared memory. We provide a concise abstract definition of communication uncertainty that is independent of the particular information transfer mechanism.

## 1. Introduction

In this paper we focus on some of the key problems of fault tolerant distributed systems and what makes them hard to solve. We offer a unified theory that allows interpretation and comparison of many of the most important known limits on distributed systems. Among the demands placed on the designer of a distributed system, there is a competition between opposing demands for safety (consistency) and timeliness. We will argue that systems that require safety are generally subject to an isolation phenomenon that prevents timely coordination. We attempt to capture at a very high level of abstraction the possibility that faults or delays may force some processes in a distributed system to make decisions and take actions that are not entirely coordinated with those of other processes that remain functioning.

Our results are obtained in an abstract model of a distributed system that consists of a set of executions of the system together with an analysis of the system presented as a family of partial executions. Requirements for such an analysis are presented axiomatically. Any family that satisfies these requirements will serve as an analysis of the system; but some families provide a richer analysis than others.

Our results could be obtained using a knowledge oriented analysis based on possible executions in place of our families of partial executions (cf. [FHV, HM, HF, CM, PT]). However, the structures in such an analysis would be much more complex than those we introduce. The surprise is that we can obtain so much without resorting to knowledge based argument, relying only on a few simple axioms and definitions.

Our theory does not require a global concept of time, depending instead on a local ordering of events at each process in the distributed system. A novel contribution of our approach is the definition of "communication uncertainty" without the explicit mention of messages or other mechanisms for information transfer (cf. [L,DDS]).

The earliest discussed glimpse at this phenomenon of communication uncertainty, and one of the most intuitive, comes in the form of the Two Generals' Problem [G] (also known as the Chinese Generals' Problem). Two generals of allied armies are encamped on opposite hills above a valley occupied by the enemy. The generals' problem is to coordinate a time of attack. However, the only way they can communicate is by messenger through the valley, so there is uncertainty about whether any particular message will get through.

One of the most studied recent impossibility results in computer science is the result of [FLP] that it is impossible to reach consensus in an asynchronous system in the presence of at most one process crash failure, even when eventual communication is guaranteed. This result was generalized in [DDS] to apply to systems with various synchrony assumptions. The consensus problem is a coordination problem in the sense that all the processes that do not fail must agree on a common output. Multivalence is introduced because each process is given an input from a set of at least two values, and if all inputs agree, then all outputs must agree with the inputs. Note the similarity to the atomic commit problem (v. [BHG,G]) in which each process has some "input" that allows it to make a unilateral decision on whether to abort or allow commit; if all inputs allow commit and there are no failures, then the agreed on output must be commit. The essential distinction between the two problems is the constraint placed on processes recovering from failure in the atomic commit problem. The consensus problem is simpler because it puts no constraints on the actions taken by processes once they have failed. In both cases, communication is assumed to be uncertain. A fundamental characteristic of a communication uncertain system is that no information is conveyed to a process about the state of other processes by the number of actions the process takes. Using a new definition of communication uncertainty that attempts to capture this characteristic, we show that communication uncertain systems are vulnerable to isolation that can block the solution of any nontrivial coordination problem including consensus. Moreover, we show that, even in a model with guaranteed eventual communication and guaranteed eventual process repair, safety requires blocking. We extend this result to a generalization of the Fischer, Lynch, Paterson impossibility result. This generalization is expressed without reference to any specific mechanism for information transfer, such as messages, and allows immediate application to systems in which information is transferred by other mechanisms including shared memory

The reader may wish to compare our approach with alternate approaches (cf. [CM, PT, BMZ]).

The remainder of the paper is organized as follows: in Section 2 we provide our formal theory of distributed systems and give a general result relating safety and blocking for systems that solve nontrivial coordination problems; in Section 3 we provide a methodology for modelling distributed systems and prove the promised generalization of the [FLP] result.

## 2. Formal Model of Coordination.

We consider systems composed of at least two independently acting processes that can communicate with each other. The set of all possible process actions is denoted $\mathcal{A}$. During an execution of a system, each process has an infinite ($\omega$ ordered) sequence of events called the *local event sequence*. Each *event* $e$ is composed of (1) an *id*, $id(e)$, consisting of the location at which the event takes place $@(e) \in \mathcal{P}$ (we identify a process with the location at which the events of its local event sequence take place), and a positive integer *sequence number* $s(e)$, (2) an *action*, $act(e) \in \mathcal{A}$, and (3) a *failure bit*, $f(e)$. Thus, in any execution, $s$ is a function from events to positive integers, @ is a function from events to processes, and $f$ is a function from events to $\{0,1\}$. An event $e$ with $f(e) = 1$ is called a *failure event*. The function @ applied to any set of events will be the set of processes at which the events take place.

Formally, an *execution* is a set of events, such that projection on ids is a one-one mapping onto the Cartesian product of $\mathcal{P}$ (the set of processes) and the positive integers. We define a *distributed system* as a set of executions with at least two processes. We will use the term *system* as synonymous with the term *distributed system.*

A set of events is said to be *compatible* with respect to a system if it is contained in one of the executions of the system. When the particular system is clear from the context we will use the term "compatible" without mentioning the system. Two sets of events are said to be *compatible* if their union is compatible.

A *prefix* is a finite compatible set of events that is *closed under precedence* (if prefix $x$ contains event $e$, then it contains all events at $@(e)$ with smaller sequence numbers than $s(e)$). A prefix may be thought of as a finite partial execution. We use incompatibilities between prefixes to capture the idea of information transfer within a distributed system.

A *coordination problem* for a distributed system consists of a subset $\mathcal{O}$ of $\mathcal{A}$ called *output actions*, a subset $\mathcal{R}$ of $\mathcal{P}$ called *participants* with cardinality at least two, and an equivalence relation $\equiv$ on $\mathcal{O}$ called *agreement.* A distributed system is said to *have a protocol* for a coordination problem $< \mathcal{O}, \mathcal{R}, \equiv>$ if, in every execution of the system, each process in $\mathcal{R}$ either fails or has at most one action from $\mathcal{O}$. (Formally, a *protocol* is a system together with a particular coordination problem for the system such that participants behave as described above in each execution.) A protocol is said to be *total* if, in each execution each process in $\mathcal{R}$ either fails or has exactly one action from $\mathcal{O}$. A protocol is said to be *safe* if, in each execution, all actions from $\mathcal{O}$ taken by processes in $\mathcal{R}$ that have not failed agree with respect to $\equiv$. When two output actions do not agree we will say they *disagree.* A coordination problem is said to be *nontrivial* for a distributed system if there are disagreeing output actions in the set of failure free executions of the system (the disagreeing outputs may be in different executions). Examples of nontrivial coordination problems include the atomic commit problem, the consensus problem, and the atomic broadcast problem. (For atomic broadcast, one participant is given an input value. If this participant does not fail, then all participants that do not fail must give the input value as output. In any case, all participants that do not fail must give the same value as output. Thus failure free executions of a protocol for the atomic broadcast problem must have disagreeing outputs if their inputs disagree.)

Our goal is to analyze our distributed system by choosing some family of prefixes to represent the "global states" or "cuts." The intuition leading to our axiomatization of such a family is contained in an example that assumes a Newtonian framework of real time. If each event in the system has associated with it a real time, and if events at one process can only "cause" or "influence" later events at other processes (no simultaneous action), then each prefix that corresponds to events that happen before some real time corresponds to a global state. We also add any subset of the events that happen at some particular real time to those that happen before. For each execution, the prefixes in the family are partially ordered by real time. This family of prefixes satisfies the first three axioms discussed below: Closure, Compatibility, and Separability. It turns out that for many purposes, we can drop the assumption of real time and simultaneity and obtain the same results by constructing another family of prefixes that satisfies the first two of these axioms. Moreover, we can characterize the intutive concept of "communication uncertainty" with the second pair of axioms below: Separability and Autonomy.

If $x$ and $y$ are sets of events, we denote the union of $x$ and $y$ by $x + y$; and, the set of events in $x$ but not in $y$ by $x - y$. Thus $@(y - x)$ is the set of locations of events in $y$ but not in $x$, which may not be the same as $@(y) - @(x)$. A family of sets of events is said to be *closed under compatible union and intersection* if, whenever $x$ and $y$ are in the family and $x + y$ is compatible, then $x + y$ and $x \cap y$ are in the family. A family of sets of events is said to *cover* the set of prefixes of a distributed system if, for each prefix $x$ and execution $z$, if $x$ is contained in $z$, then there is a member $y$ of the family that contains $x$ and is contained in $z$. Prefix $y$ is said to be a *failure conserving* extension of prefix $x$ if the only failure events in $y - x$ occur at processes that failed in $x$. A failure conserving extension $y$ of $x$ is a *strict* extension if the failure bit of any event in $y - x$ is the same as the failure bit of the last event in $x$ at the same process.

An *analysis* of a distributed system is a family of prefixes (called *cuts*) that satisfies the following two axioms:
**Closure Axiom:** The family contains the empty prefix, covers the set of prefixes of the system, and is closed under compatible union and compatible intersection.
**Compatibility Axiom:** If $x$, $y$, and $z$ are cuts with $x \subset y$ and $x \subset z$, and $@(y - x)$ and $@(z - x)$ are disjoint, then $y$ and $z$ are compatible.

The cuts are to be viewed as global states of our system; however, in general in our model, there is no notion of time or simultaneity, so cuts should not be considered to represent any notion of concurrency. Every system possesses at least one analysis. If a system possesses an analysis satisfying the following two additional axioms then we say the system has *communication uncertainty:*
**Separability Axiom:** For any pair of events at different processes that are contained in a cut, there is a compatible cut that contains one event but not the other.
**Autonomy Axiom:** For each cut $x$ and process $p$, there is a cut $y$ containing $x$ such that $@(y - x)$ contains only $p$ and $y$ is a failure conserving extension of $x$.

It is communication uncertainty rather than any particular form of asynchrony that leads to impossibility results like those of [FLP]. Let the asynchronous message based systems of [FLP] (with no restriction on failure types or numbers) with analysis consisting of the family of prefixes that contain the corresponding message send event for every contained message receive event be called *standard message based systems.* It is easy to check that this analysis satisfies all four axioms above. (cf. [L]).

The results in the remainder of this section assume communication uncertainty. In the next section, an analysis that satisfies the separability axiom will be called *separable* and an analysis that satisfies the autonomy axiom will be called *autonomous.*

Cut $y$ is said to be a *successor* of set of events $x$ when $x \subset y$ and there is no cut $z$ such that $x \subset z \subset y$. The transitive closure of the successor relation is denoted *leads to.* When $x$ leads to $y$, $y$ is said to *extend* $x$. When $x$ leads to $y$ and $|@(y - x)| = 1$, we say $x$ leads to $y$ *locally* and $y$ is a *local extension* of $x$. Note that a successor to a set of events must be a cut and therefore a prefix.

If cut $x$ has incompatible successor cuts $y$ and $z$, then the triple $< x, y, z >$ is said to be a *fork.* The fork $< x, y, z >$ is said to be *local* if $@(y - x) \cup @(z - x)$ consists of one process. Fork $< x, y, z >$ is said to be *failure free* if $y$ and $z$ are failure free.

**Lemma 2.1.:** If cut $y$ is a successor of cut $x$, then there is exactly one process in $@(y-x)$.

**Proof:** Suppose cut $y$ were a successor of cut $x$ and $@(y-x)$ contained more than one process. By the separability axiom, since $y-x$ contains events at different processes, there is a cut $z$ compatible with $y$ that contains some event of $y-x$ but does not contain some other event of $y-x$. By the closure axiom, the intersection of $y$ and $z$ is a cut properly contained in $y$ but containing some event of $y-x$. Now $x$ is also a cut properly contained in $y$, so the union $w$ of $x$ with the intersection of $y$ and $z$ is a compatible set of events and thus a cut. But $w$ properly contains $x$ and is properly contained in y; so $y$ could not be a successor of $x$.

■

**Lemma 2.2.:** Every fork is local.

The proof is omitted.

A sequence $\{x(i)\}$ of prefixes is called an *ascending chain* if, for each $i$, $x(i) \subset x(i+1)$. A protocol is called *blocking* if there is an infinite ascending chain $\{x(i)\}$ such that some participant takes infinitely many actions in $\cup\{x(i)\}$ without failing or giving output. We say a protocol is *timely* if there exists an integer $k$ such that, in every execution, every participant either fails or gives an output before it has taken $k$ actions. Thus if a protocol is blocking, then it is not timely (cf. "wait-free" in [H]). Moreover, if a protocol is timely, then it is total.

Consider some fixed protocol. A cut is said to be *multivalent* if it is contained in failure free executions with disagreeing outputs. If a cut is contained in some failure free execution and if all outputs in the failure free executions that contain it agree, then the cut is said to be *univalent*. Note that if a system has a nontrivial coordination problem, then the empty cut is multivalent.

**Theorem 2.3.:** Any safe protocol for a nontrivial coordination problem in a communication uncertain system must be blocking.

**Proof Sketch:** The empty cut is multivalent. Either there is an infinite ascending chain of multivalent cuts or there is a multivalent cut with no multivalent successors. If there is an infinite ascending chain of multivalent cuts, then the protocol is blocking. If there is a multivalent cut $x$ with no multivalent successors, then there must be a failure free fork $< x, y, z >$. By Lemma 2.2, there is a process $p$ not contained in $@(y-x) \cup @(z-x)$. Now repeated application of the autonomy axiom yields an infinite ascending chain of failure free cuts, beginning with $x$, in which only $p$ takes actions after $x$. By the compatibility axiom, each of these cuts must be compatible with both $y$ and $z$, so $p$ can neither fail nor give output in any of them. Thus, in any case, the protocol is blocking.

■

**Corollary 2.4.:** There is no safe timely protocol for a nontrivial coordination problem in a communication uncertain system.

## 3. Production of Models of Distributed Systems by Restriction

### 3.1. Syntax and Environment

Our method for constructing models of systems starts with a syntactically defined free algebra of executions. By restriction we introduce what we call an environment. This is a set of executions that contains in some sense executions for all possible protocols that could be written for the system. Then we further restrict the set of executions to those that correspond to the actions (state transitions) of a fixed set of automata. All our restrictions are specified as properties of sets of events, independent of any consideration about what system is being restricted. Thus an execution either satisfies or does not satisfy a restriction, independent of the system from which it is taken. Formally, a *restriction* is a property of sets of events. Applying a restriction $\rho$ to a system $S$ produces the set of executions $\rho S$ that satisfy the property $\rho$.

To model each example, we start with what we call a *syntactic* system consisting of all executions that can be formed from given sets of processes and actions. we apply restrictions to the syntactic system to obtain an environment, where an *environment* is a system $E$ such that, for any execution of $E$, changing any failure bit produces another execution of $E$. Note that under this definition, a syntactic system is an environment. A restriction is said to be *environmental* in system $S$ if the restriction is independent of which events are failure events for executions in $S$. For environments, the failure bits of events are irrelevant. We use the failure bits of events in a system to indicate events that are not according to the specifications of the protocols of the system but are still possible within an environment that contains the system. Note that if $E$ is an environment and $\rho$ is an environmental restriction in $E$, then $\rho E$ is also an environment.

We can think of an environment as containing many potential protocols. To specify a single protocol, we restrict the environment to a system in which the actions correspond to state transitions of a given set of automata and the events with actions not specified by the automata are the failure events (cf. IO automata [LM,LT]).

### 3.2. Completeness and Continuity

A system is said to be *complete* if the union of the events of any ascending chain of prefixes is contained in an execution. Note that a complete system cannot have a total blocking protocol for a coordination problem.

An example of a restriction that does not preserve completeness is *eventual message delivery*. This restriction holds of a set of events if for each message sent the set contains a corresponding receive event or a failure event at the target process. Thus, if a system has eventual message delivery, then an infinite ascending chain of cuts in which some message is never delivered while its target takes infinitely many actions without failing is not contained in any execution of the system.

The following result follows immediately from Theorem 2.3.

**Corollary 3.1.:** (Generalization of Two Generals Problem) There is no safe total protocol for a nontrivial coordination problem in a complete communication uncertain system.

For each system $S$ and restriction $\rho$, we define the restriction $ext_S(\rho)$ to hold of a set of events if that set can be extended to an execution of $S$ that satisfies $\rho$. In other words, $ext_S(\rho)(x)$ if, and only if, $x$ is a compatible set of events in $\rho S$. A restriction $\rho$ is said to be *continuous* on system $S$ if the restriction $ext_S(\rho)$ is inherited by the union of any ascending chain of prefixes for which it holds. The following lemmas are immediate from the definitions.

**Lemma 3.2.:** System $\rho S$ is complete if, and only if, $\rho$ is continuous on $S$.

**Lemma 3.3.:** If restrictions $\rho$ and $\sigma$ are continuous on system $S$ then $(\rho|\sigma)$ is continuous on $S$. If, in addition, $ext_S(\rho) \wedge ext_S(\sigma)$ implies $ext_S(\rho \wedge \sigma)$, then $(\rho \wedge \sigma)$ is also continuous on $S$.

If prefix $z$ can be obtained from prefix $y$ by changing only failure bits, then $z$ and $y$ are said to be *similar.* A restriction $\rho$ is said to *impose a protocol* on system $S$ if for every prefix or execution $y$ of $S$, there is exactly one prefix or execution of $\rho S$ that is similar to $y$.

**Lemma 3.4.:** If restriction $\rho$ imposes a protocol on complete system $S$, then $\rho$ is continuous on $S$.

The proof is omitted.

When we restrict an environment to executions with failure bits determined by a given set of automata (with state transitions corresponding to the actions of the environment), the restriction imposes a protocol on the environment. Thus such a restriction preserves completeness.

## 3.3. Special Restriction Types

We focus on two types of restrictions and the system properties that they preserve. The types are called "pattern enforcing" and "coherent." Each type is defined in this section.

If $\rho$ is a restriction and $R$ is a set of processes, then $\rho$ is said to have *base* $R$ if $\rho$ is determined by the projection of its argument on the set of events that take place at processes in $R$. Here we define a specific type of restriction that can be used to state a

powerful result for uncertain systems. Restriction $\rho$ is said to be *basic* if
(1) it is inherited by supersets,
(2) it is inherited by some finite subset, and
(3) it has a base other than the set of all processes.

Messages provide a special case of what we call a basic pattern. The pair $< \alpha, \beta >$ of basic restrictions is a *basic pattern* for an analysis $A$ if
(1) for each $x \in A$, $\beta(x)$ implies $\alpha(x)$;
(2) for each $x \in A$, $\alpha(x)$ implies $x$ has a failure conserving local extension $y$ in $A$ such that $\beta(y)$; and
(3) there is a base for $\beta$ consisting of a single process.

When a restriction can be expressed as the countable conjunction of implications of the form $(\alpha \Rightarrow \beta)$, where $< \alpha, \beta >$ forms a basic pattern for analysis $A$, we call the restriction *pattern enforcing* with respect to $A$. If "failure conserving" is replaced by "strict" in (2) above, then we call the restriction *strict pattern enforcing* with respect to $A$.

The restriction to "eventual message delivery" is a strict pattern enforcing restriction for standard message based systems. Each basic pattern $< \alpha, \beta >$ corresponds to the send and receive of a particular message. In general restrictions that specify eventual communication are usually pattern enforcing. However, restrictions that specify communication within a bounded number of actions (e.g. if the communication is initiated before action $k$ of the sender, then receipt must take place before action $2k$ of the receiver) are usually not pattern enforcing because they cannot be expressed in terms of patterns that satisfy (2) above.

**Lemma 3.5.:** If $S$ is complete and $\rho$ is pattern enforcing with respect to some analysis of $S$, then any analysis for $S$ forms an analysis for $\rho S$. Moreover, any separable or autonomous analysis for $S$ forms a respectively separable or autonomous analysis for $\rho S$.

The proof is omitted.

A property $\rho$ is said to be *pattern enforcing* for a system if it is pattern enforcing for some analysis of the system.

**Corollary 3.6.:** If restriction $\rho$ is pattern enforcing for a complete communication uncertain system $S$, then $\rho S$ is communication uncertain.

One very strong restriction that is sometimes assumed in modelling distributed systems is the property of *eventual repair with immunity to failure* (after some point each process that has failed is repaired and has no subsequent failure). A weaker alternative that is somewhat more realistic is what we call *eventual repair* (for any integer $k$ and any process $p$, there is some point after which $p$ takes $k$ actions without failing). The restriction to "eventual repair" is a pattern enforcing restriction for standard message based systems. For each basic pattern $< \alpha, \beta >$, we take $\alpha \equiv$ **true**. For each process

$p$ and integer $k$ there is a corresponding $\beta$ that requires $p$ to have at least $k$ contiguous events that are not failures in its local event sequence.

A restriction $\rho$ is said to be *coherent* for analysis $A$ of system $S$ if it satisfies the following conditions:
(1) the empty prefix satisfies $ext_S(\rho)$,
(2) $ext_S(\rho)$ is inherited by compatible unions of members of $A$, and
(3) $ext_S(\rho)$ is inherited by strict extensions.
We say a restriction is *coherent* for a system if it is coherent for some analysis for the system.

If $A$ is an analysis for system $S$ and $\rho$ is a restriction, then we define $\rho A$ to be the family of prefixes of $\rho S$ that are members of $A$.

**Lemma 3.7. :** If restriction $\rho$ is coherent for analysis $A$ of system $S$ then $\rho A$ forms an analysis for $\rho S$. Moreover, $\rho$ preserves separability and autonomy.

The proof is omitted.

Let $\mathcal{R}$ be any subset of the set of processes. Let $\rho$ be the restriction that is satisfied when failure events occur only at $\mathcal{R}$. Let $S$ be any system that contains an execution satisfying $\rho$. Then $\rho$ is coherent for $S$.

**Theorem 3.8. :** Suppose A is a separable autonomous analysis for a complete system S. Let $\rho$ be any pattern enforcing restriction for A in S. Let $\tau$ be the restriction of having no process failures. Then $(\rho \wedge \tau)S$ has communication uncertainty.

**Proof Sketch:** Since $\tau$ is coherent, $\tau A$ forms a separable autonomous analysis for $\tau S$ by Lemma 3.7. Note that $\tau$ is continuous on $S$ so that $\tau S$ complete. Thus by Lemma 3.5 it suffices to show that $\rho$ is pattern enforcing for $\tau A$ in $\tau S$. But this is straightforward since a failure conserving extension of a failure free cut must also be failure free.
■

**Corollary 3.9.:** Let $S$ be the restriction of a standard message based system to executions with any combination of eventual message delivery, eventual repair, and no process failures. Then $S$ has no safe timely protocol for a nontrivial coordination problem.

**Proof Sketch:** The Corollary follows from Corollary 2.4 and Theorem 3.8 for the conjunction of all three restrictions.
■

**Lemma 3.10.:** If $\rho$ is a strict pattern enforcing restriction for analysis $A$ of system $S$ and $\sigma$ is a coherent restriction for $A$, then $\rho$ is a strict pattern enforcing restriction for analysis $\sigma A$ of system $\sigma S$.

The proof is omitted.

## 3.4. Crash Failures

A process is said to suffer a *crash failure* with respect to analysis $A$ if its local sequence of events consists of failure free events followed by an infinite number of special events called *stopped* events such that if $a$ is some last stopped event in a member $x$ of $A$, then either the removal of $a$ or the addition of a next stopped event provides another member of $A$. A system is said to be *subject to crash failures* with respect to analysis $A$ if, for each process $p$, any prefix $x$ is contained in a prefix $y$ such that $y - x$ is a stopped event at $p$ with respect to $A$. Note that the restriction of having only crash failures is continuous on any complete system that is subject to crash failures. Also, having only crash failures is inherited by failure conserving extensions in any system that is subject to crash failures and has only stopped events as failure events.

**Theorem 3.11. :** Suppose $A$ is a separable autonomous analysis for a complete system $S$ such that $S$ is subject to crash failures and has only stopped events as failure events with respect to $A$. Let $\rho$ be any pattern enforcing restriction for $A$ in $S$; let $\sigma$ be the restriction of having only crash failures; let $\tau$ be the restriction of having at most one failure; and let $\pi$ be the conjuction $\rho \wedge \sigma \wedge \tau$. Then $\pi S$ is a system that contains no safe total protocol for a nontrivial coordination problem.

**Proof Sketch:** (This proof uses techniques adapted from [FLP].) Assume the hypotheses of the theorem and assume that the conclusion is not satisfied because $\pi S$ has a safe total protocol for a nontrivial coordination problem. We will write $A'$ for $(\rho \wedge \sigma)A$ and $S'$ for $(\rho \wedge \sigma)S$. By Lemmas 3.5, 3.7, and 3.10, $A'$ is a separable autonomous analysis of $S'$.

Let $\mu(p)$ be the restriction of having failure events only at process $p$. Let $\nu$ be the restriction of having no failure events. Then $\mu(p)$ and $\nu$ are both coherent restrictions for $A'$ in $S'$. Thus by Lemma 3.7, $\mu(p)A'$ and $\nu A'$ are separable autonomous analyses of their respective systems.

The triple $< x, y, z >$ of members of $A'$ will be said to form a *local disagreement* if $x$ is multivalent and leads to disagreeing univalent $y$ and $z$ with only one process in $@(y - x) \cup @(z - x)$. We will show that $S'$ can have no local disagreements. Suppose that $< x, y, z >$ were a local disagreement at process $p$. Since they are all failure free, $x$, $y$, and $z$ are all prefixes of $\mu(p)S'$. Since $\mu(p)S'$ is a subset of the executions of $\pi S$, each execution of $\mu(p)S'$ must contain output for all processes except possibly $p$. There must be some execution of $\mu(p)S'$ that contains $x$ and in which $p$ crashes immediately after its last event in $x$. For this execution, some process must give an output; so there must be some member of $\mu(p)A'$ that is univalent and has no event at $p$ except for those of $x$. Since $\mu(p)A'$ forms an analysis for $\mu(p)S'$, this univalent cut must be compatible with both $y$ and $z$, contradicting the safety of the hypothesized protocol. Thus $S'$ can have no local disagreements.

Since $S$ has a total protocol for a nontrivial coordination problem, the empty cut of $\nu A'$ is multivalent in $\nu S'$. We can now construct an infinite ascending chain of multivalent cuts in which each process takes infinitely many actions without failing. The construction iterates $n+1$ stages where $n$ is the number of processes. In stages that are $i \bmod (n+1)$ with $0 < i < n+1$, we construct a cut containing the cut we have together with some new action of process $i$. In stages that are $0 \bmod (n+1)$, if $\rho$ holds of the cut we have, then we do nothing; otherwise, we construct a failure free containing cut that satisfies the first implication $\alpha_i \Rightarrow \beta_i$ not satisfied by the original cut.

Using the key technique of [FLP], it is straightforward to show that if, at some stage, we cannot produce a multivalent cut with the required property, then there must be a local disagreement. Thus we can construct the desired infinite ascending chain. By construction the chain must satisfy property $\rho$. Since the chain has infinitely many actions at each process, its union is a failure free execution in $\pi S$. But this contradicts the assumption that the protocol is total in $\pi S$.
■

**Corollary 3.12. :** (FLP) Let $S$ be the restriction of a standard message based system to executions with eventual message delivery, no failures except crash failures, and at most one process failure. Then $S$ has no safe total protocol for a nontrivial coordination problem.

## 4. Conclusion.

When communication is uncertain, isolation is possible and there is no way to guarantee safety and timely (nontrivial) coordination among distributed processes. Usually the probability of isolation is small even though some components may be inoperative for long periods of time. Blocking prevents transient failures from causing a loss of consistency during isolation. Much as we would like a truly nonblocking total coordination protocol, Corollaries 3.1 and 3.6 imply that neither eventual message delivery nor eventual repair is enough to guarantee safety without blocking in an uncertain environment. We suggest that rather than search for real systems without the problems of communication uncertainty, we should search for systems in which isolation has low probability and choose either consistency or timeliness to guarantee, relegating the other goal to highly probable but not certain.

## 5. Acknowledgements

The authors would like to thank Joe Halpern for helpful comments on an earlier version of this work.

## References.

[BHG] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrent Control and Recovery in Database Systems,* Addison-Weseley, 1987.

[BMZ] O. Biran, S. Moran, and S. Zaks, "A Combinatorial Characterization of the Distributed Tasks Which are Solvable in the Presence of one Faulty Processor." *Proc. 7nd ACM Symp. on PODC*, (1988) 263-273.

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

[DDS] D. Dolev, C. Dwork, and L. Stockmeyer, "On the Minimal Synchronism Needed for Distributed Consensus," *Journal of the ACM*, (1987) 34:77-97.

[FHV] R. Fagin, J. Y. Halpern, M. Y. Vardi, "A Model-Theoretic Analysis of Knowledge," *Proc. 25th IEEE Symp. on Foundation of Computer Science*, (1984) 268-278.

[FLP] M. J. Fischer, N. A. Lynch and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *JACM 32* (1985) 373-382.

[G] J. N. Gray, "Notes on Database Operating Systems," *Operating Systems: an advanced course, Lecture Notes in Computer Science 60*, Springer Verlag (1978) 393-481.

[HF] J. Halpern and R. Fagin, "A Formal Model of Knowledge, Action, and Communication in Distributed Systems," *Proc. 4th ACM Symp. on PODC*, (1985) 224-236.

[HM] J. Y. Halpern, and Y. Moses, "Knowledge and Common Knowledge in a Distributed Environment," *Proc. 3rd ACM Symp. on PODC*, (1984).

[H] Maurice P. Herlihy, "Impossibility and Universality Results of Wait-Free Synchronization," *Proc. 7nd ACM Symp. on PODC*, (1988) 276-290.

[L] L. Lamport, "Time Clocks and the Ordering of Events in a Distributed System," *CACM 21*,7 (1978) 558-565.

[LM] N. A. Lynch, and M. Merritt, "Introduction to the Theory of Nested Transactions," *International Conf. on Database Theory*, Sep. 1986, Rome, Italy, 278-305. Technical Report MIT/LCS/TR-367, MIT, Lab. for Computer Science, April 1986.

[LT] N. A. Lynch, and M. R. Tuttle, *Hierarchical Correctness Proofs for Distributed Algorithms*, Technical Report MIT/LCS/TR-387, MIT, Lab. for Computer Science, April 1987.

[PT] P. Panangaden and K. Taylor, "Concurrent Common Knowledge: A New Definition of Agreement for Asynchronous Systems," *Proc. 7nd ACM Symp. on PODC*, (1988) 197-210.