International Journal of Parallel Programming, Vol. 18, No. 4, 1989

Initial Failures in Distributed Computations

Gadi Taubenfeld,[1,2] Shmuel Katz,[1] and Shlomo Moran[1,3]

Received August 1988; Revised January 1990

We investigate the possibility of solving problems in completely asynchronous message passing systems where a number of processes may fail prior to execution. By using game-theoretical notions, necessary and sufficient conditions are provided for solving problems is such a model with an without a termination requirement. An upper bound on the message complexity for solving any problem in the model is given, as well as a simple design concept for constructing a solution to any solvable problem.

KEY WORDS: Asynchronous protocols; crash failures; initial failures; winning strategy.

1. INTRODUCTION

In this paper we investigate both the possibility and the impossibility of solving certain problems in an unreliable, completely asynchronous, distributed system where undetectable initial failures may occur. Initial failures are a very weak type of failure where processes may fail only prior to the execution and no event can happen on a process after it fails. Thus it is assumed that any process that begins executing will not fail, but some processes may never begin executing. Initial failures may occur in situations such as recovery from a breakdown of a network. Such initial failures are a special case of *crash* (fail-stop) failures in which a process may become

[1] Computer Science Department, Technion, Haifa 32000, Israel.

[2] Supported in part by the Guttwirth Fellowship, by the National Science Foundation under grant CCR-8405478, and by the Hebrew Technical Institute scholarship. Current address: Computer Science Department Yale University, New Haven, Connecticut 06520.

[3] Supported in part by Technion V.P.R. fund—C. Wellner Research fund.

faulty at any time during an execution. Obviously, if a protocol cannot tolerate initial failures then it cannot tolerate crash failures but not necessarily vice versa.

We distinguish between the case when the protocal is not required to terminate after producing the correct result, and the case when the protocol is required to terminate. For each of these cases we provide necessary and sufficient conditions for the solvability of problems. As we shall see, requiring termination strictly reduces the class of solvable problems. The conditions are presented by introducing simple games and relating the possibility to win in these games to the possibility of solving problems.

The complexity of protocols designed to solve certain problems is greatly affected by whatever assumption is made about the system's reliability. For the initial failure model we provide an upper bound on the message complexity for solving any problem by showing that the message complexity of any solvable problem is bounded by the message complexity for solving the leader election problem.

Several protocols in the literature have been designed to operate properly in an environment where initial failures may occur. Fischer *et al.*(1) presented a protocol that solves the consensus problem (*i.e.* where all processes decide on the same value) and tolerates initial failures of up to (not including) half of the processes. Protocols for leader election and spanning tree construction that also tolerate initial failures of up to half of the processes were designed in Ref. 2. A solution to the leader election problem appeared also in Ref. 3. Throughout the paper we will assume that the number of faulty processes is always less than half of the total number of processes.

There has been extensive investigation into the nature of asynchronous systems with the stronger assumption that undetectable crash failures may occur. It is proven in Ref. 1 that in asynchronous systems there cannot exist a nontrivial consensus protocol that tolerates even a single process crash failure. This fundamental result has been extended to other models of computation.(4,5) Various extensions,(6–8) also for a single fault, prove the impossibility of other problems such as choosing a leader, ranking and constructing a spanning tree using several new techniques. Other recent works point out some specific problems that can be solved in asynchronous systems with numerous faulty processes, and prove impossibility results for a hierarchy of problems assuming up to t crash failures.(9–11) (As will be seen later, in our terminology, the solutions given in Ref. 9 are non-terminating.) Recently, a complete combinatorial characterizations, for the solvability of problems in asynchronous shared memory and message passing models where crash failures may occur using *randomized protocols* was given in Ref. 12.

It appears that the necessary and sufficient condition which we give for initial failures assuming only deterministic protocols and without the termination requirement, is the same as the complete characterization which is given in Ref. 12 for crash failures in a shared memory model assuming randomized protocols. Moreover, the necessary and sufficient condition which we give here for initial failures assuming only deterministic protocols and *with* the termination requirement, is the same as the complete characterization which is given in Ref. 12 for crash failures in a message passing model assuming randomized protocols. An interesting result that follows from the similarities between these characterizations is that in a message passing model with the termination requirement, a problem can be solved by a *deterministic* protocol that can tolerate up to *t initial* failures if and only if the problem can be solved by a *randomized* protocol that can tolerate up to *t crash* failures. A similar result holds for a shared memory model when termination is not required.(13)

The rest of the paper is organized as follows. In Section 2, the notions of a problem and a protocol are defined. Section 3 describes the essential properties of asynchronous message passing systems. Sections 2 and 3 also appear, with minor changes, in Ref. 11. In Section 4, the notions of initial failure and termination are precisely defined. Section 5 introduces a two player game and it is proved that the possibility of winning this game is related to the possibility of solving problems in an environment where processes may initially fail. Sections 6 and 7 include two modified games designed to enable reasoning about the possibility of solving problems under a termination requirement. Some concluding remarks appear in Section 8.

2. DEFINITIONS AND BASIC NOTATIONS

First, the type of problems we consider is described. Let I and D be sets of input values and decision (output) values, respectively. Let n be the number of processes, and let $\bar{I}$ and $\bar{D}$ be subsets of I^n and D^n, respectively. A problem T is a mapping $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ from each n-tuple in $\bar{I}$ to subsets of n-tuples in $\bar{D}$. We call $\mathbf{a} = (a_1,..., a_n)$ where $\mathbf{a} \in \bar{I}$ an input vector, and $\mathbf{d} = (d_1,..., d_n)$ where $\mathbf{d} \in \bar{D}$, a decision vector. For such a pair of vectors, we say that a_i is the input value of process p_i, and d_i is the decision value of process p_i.

Following are some examples of problems, which we will also refer to later in the paper (the input vectors for all problems are from I^n for an arbitrary set I): (1) The *consensus* problem, where all processes must decide on the same value from an arbitrary set D; (2) The *transaction commitment* problem, where $I = D = \{0, 1\}$, and all processes decide on "1" if the input of every process is "1", and otherwise all processes decide on "0."; (3) The (leader) *election* problem, where exactly one process decides on a distinguished value from an arbitrary set D; and (4) The *sorting* problem, where all processes have input values and each process p_i decides on a value identical to the ith smallest input value.

A *protocol* is a nonempty set C of *computations* and a set $N \equiv \{p_1, ..., p_n\}$ of process id's (abbv. processes). A computation is a *finite* sequence of events. There are four types of events: *send*, *receive*, *input*, and *decide*. A *send* event, denoted $([send, m, p_k], p_i)$, represents sending a message m to process p_k by process p_i. A *receive* event, $([receive, m], p_k)$, represents receiving a message m by process p_k. An *input* event, $([input, a], p_i)$, represents reading an input value a by process p_i. A *decide* event, $([decide, d], p_i)$, represents deciding on a decision value d by process p_i. One may also consider an *internal* event in with a process executes some local computation; however nowhere else in this paper do we need to refer to such an event. We use the notation (e, p_i) to denote an arbitrary event, which may be an instance of any of the above types of events. For an event (e, p_i) we say that it occurred *on* process p_i. An event is *in* a computation iff it is one of the events in the computation sequence.

In the rest of this paper Q denotes a set of processes where $Q \subseteq N$, and $|Q|$ is the cardinality of Q. The symbols x, y, z denote computations. Also $\langle x; y \rangle$ is the sequence obtained by concatenating the two sequences x and y. An *extension* of a computation x is a computation of which x is a prefix. For an extension y of x, $(y - x)$ denotes the suffix of y obtained by removing x from y. For any x and p_i, let x_i be the subsequence of x containing all events in x that are on process p_i.

Definition. *Computation y includes x* iff x_i is a prefix of y_i for all p_i (the relation *includes* is a generalization of *extension*).

We assume that all events are unique and all messages are distinguished (*i.e.*, the same event cannot occur twice in a computation). An event $([receive, m], p_k)$ is the *complement* of the event $([send, m, p_k], p_i)$ in a computation x iff both events are in x. An event $([send, m, p_k], p_i)$ is *fulfilled* in a computation x if it is in x and its complement event $([receive, m], p_i]$ is also in x. That is, the message m sent from process p_i to process p_k has already arrived. An event $([send, m, p_k], p_i)$ is *unfulfilled* in a computation x if it is in x, and it is not *fulfilled* in x.

Definition. Computations x and y are *equivalent* w.r.t. p_i, denoted by $x \overset{i}{\sim} y$, iff $x_i = y_i$.

Note that $\overset{i}{\sim}$ is an equivalence relation over computations. For a computation x and process p_i, we defined the extensions of x that only have events on p_i.

Definition. *Extensions* $(x, i) \equiv \{y \mid y$ is an extension of x and $x \sim^j y$ for all $j = 1 \cdots i-1, i+1 \cdots n\}$.

Process p_i *reads* input a in a computation x iff the input event $([input, a], p_i)$ is in x. Process p_i *decides* on d in a computation x iff the decision event $([decide, d], p_i)$ is in x. Since we are interested in protocols that solve the type of problems mentioned previously, without loss of generality, we can assume that a process may decide only once. More formally, for any computation x and for any process p_i there is at most one input event, and at most one decision event on p_i in x.

A *protocol* $P \equiv (C, N)$ *solves* a problem $T: \bar{I} \to 2^{\bar{D}} - \{\varnothing\}$ iff (1) For each input vector $\mathbf{a} \in \bar{I}$, there exists a computation with $\mathbf{a}$ as input; (2) For every computation $z \in C$ such that in z processes $p_1, \ldots, p_n$ read input values $a_1, \ldots, a_n$ and decide on $d_1, \ldots, d_n$, if $\mathbf{a} \in \bar{I}$ then $\mathbf{d} \in T(\mathbf{a})$; and (3) In any "sufficiently long" computation on input in $\bar{I}$ all processes decide (this requirement is to be defined precisely later). Note that condition (2) restricts those computations with every process executing events (since all at least read their input), but does not relate to computations where some process has no associated events. Such computations will later be defined as those with initial failures.

We define when a set of input events is consistent. Intuitively, this is the case when all the input events in the set can happen in the same computation. Let P be a protocol that solves a problem $T: \bar{I} \to 2^{\bar{D}} - \{\varnothing\}$. For any input vector $\mathbf{a} \in \bar{I}$, the set $\{([input, a_1], p_1), \ldots, ([input, a_n], p_n)\}$ is a consistent set of input events (w.r.t. T); and any subset of a consistent set of input events is also consistent. For simplicity of presentation, we assume that in any given computation the set of input events is consistent. Throughout the paper we consider a single protocol, $P \equiv (C, N)$ that solves a problem $T: \bar{I} \to 2^{\bar{D}} - \{\varnothing\}$, and all references are made to that protocol.

3. ASYNCHRONOUS ENVIRONMENT

In this section asynchronous message passing systems are characterized by stating five properties that any protocol operating in such an environment should satisfy. The formal description of asynchronous message passing systems considered here is inspired by Chandy and Misra.(14,15)

Definition. An *asynchronous protocol* is a protocol that satisfies the following properties,

P1: Every prefix of a computation is a computation.

P2: Let $\langle x; (e, p_i)\rangle$ be a computation where (e, p_i) is not an input event, and let y be a computation that includes x such that $x \overset{i}{\sim} y$; then, $\langle y; (e, p_i)\rangle$ is a computation.

P3: For any computation x, any process p_i and any input value a; if the set of all input events in x together with the input event $([input, a], p_i)$ is consistent then there exists an extension y in $Extensions(x, i)$ such that $([input, a], p_i)$ appears in y.

P4: For an *unfulfilled* event $([send, m, p_k], p_i)$ in a computation x, there exists an extension y of x such that $y \in Extensions(x, k)$, and $([send, m, p_k], p_i)$ is *fulfilled* in y.

P5: For a computation x and an event $([receive, m], p_k)$, the sequence $\langle x; ([receive, m], p_k)\rangle$ is a computation only if $([receive, m], p_k)$ is the complement of some *unfulfilled* event in x.

We note that from property $P1$ the empty sequence (denoted by *null*) is a computation. Intuitively, property $P2$ means that if an event (e, p_i) can happen at a process p_i at some point in a computation, then the same event can happen at a later point, provided that it is not an input event, and p_i has taken no steps between the two points. Property $P3$ means that a process that has not yet read an input value may read any of the input values not conflicting with the input values already read by other processes. For example, if we assume that the input values different processes may read in the same computation are distinct, then a process may read any value that has not already been read by other processes. Property $P4$ means that it is always possible for a process to receive a message sent to it. Property $P5$ means that a message is received only if it was sent previously and that it cannot be received twice.

4. INITIAL FAILURES AND TERMINATION

In this section we introduce two notions. First we define a class of protocols that can tolerate multiple *initial* failures. An initial failure may occur only prior to the beginning of a process' execution and the process cannot recover. That is, once a process starts operating it is guaranteed that it will never fail. We reiterate that initial failure is a special case of *crash* failure, in which a process may become faulty at any time during an execution. We then introduce the notion of termination and define protocols that can both tolerate initial failures and terminate.

We first introduce the notion of an *enabled* process. Process p_i is *enabled* at computation x iff there exists an event (e, p_i) such that $\langle x; (e, p_i)\rangle$ is a computation. It follows from properties $P2$ and $P3$ that a process cannot become passive (*i.e.*, not enabled) as a result of an event on some other process. In order to define initial failure and termination formally, we need the concept of an *initial Q-fair sequence*. Let Q be a set of processes, an *initial Q-fair sequence* w.r.t. a given protocol is a (possibly infinite) sequence of events, where: (1) Each finite prefix is a computation; (2) For an *enabled* process p_i at some prefix x, if $p_i \in Q$ then there exists another prefix y that is an extension of x such that there is an event (e, p_i) in $(y-x)$; (3) For any send event that is *unfulfilled* in some prefix and in which a message was sent between processes of Q, the event is *fulfilled* in another prefix; (4) The sequence $\langle x; ([receive, m], p_k)\rangle$, is a prefix only if $([receive, m], p_k)$ is the complement of some *unfulfilled* event in x; and (5) The sequence consists only of events on processes belonging to Q.

An initial Q-fair sequence captures the intuition of an execution in which only processes belonging to Q participate, where all enabled processes that belong to Q can proceed, all messages sent between processes of Q are eventually delivered, and a message is received only if it was sent previously. Note that messages may be sent to processes not in Q. An initial Q-fair sequence may be *infinite* and in such a case it is not a computation. It follows from $P2$–$P5$ that, in asynchronous protocols, for every set of processes Q, any computation consisting only of events on processes belonging to Q is a prefix of an initial Q-fair sequence. The requirement that all messages are received after they are sent (requirement (4) in the definition of an initial Q-fair sequence), follows from $P5$ and requirement (1).

Informally, a protocol can tolerate up to t initial failures if in spite of a failure of any group of up to t processes at the beginning of the computation, each of the remaining processes eventually decides on some value. We now characterize such protocols formally.

Definition. A protocol *can tolerate* up to t *initial failures* iff for every set Q of processes where $|Q| \geqslant n-t$, every initial Q-fair sequence has a finite prefix in which any $p_i \in Q$ has decided.

For later reference we call a protocol that can tolerate up to t initial failures an *initial*(t) protocol. Note that for any value t such that $0 \leqslant t < n$, the class of *initial*(t) protocols includes the class of *initial*($t+1$) protocols. To see that the inclusion is strict, consider the *rotating*(t) problem, where each process p_i has to decide on a decision value from the set of input values of processes $p_{i(\text{mod}\,n)+1}, \dots, p_{i+t-1(\text{mod}\,n)+1}$. In such a *rotating*($t$) problem, if all process $p_{i(\text{mod}\,n)+1}, \dots, p_{i+t-1(\text{mod}\,n)+1}$ fail, process p_i will never be able to decide.

From this definition, it follows that for any initial(0) protocol in any "long enough" execution where no process fails (*i.e.*, an initial N-fair sequence), each process (eventually) decides on a value. In fact, this formally expresses requirement (3) from the definition of *solves* given in Section 2. Thus, any protocol that solves a problem should (by definition) at least be an initial(0) protocol.

In order to present the above notion of a protocol that can tolerate t intial failures we did not have to define the notion of a faulty process. We concentrate on the role of the correct processes in order to capture the nature of robustness. By using the notion of an initial Q-fair sequence we have described an execution in which all processes in Q are correct, and only for those processes do we require that they eventually decide. However, there is a way to define a fault tolerant protocol by first defining the notion of a faulty process as done by Hadzilacos.(16) This involves the introduction of an additional type of event, which signals the fact that a process is faulty. Our approach seems so be more suitable for the model under consideration, since it captures the fact that in systems where a failure of a process is not detectable, a faulty process cannot be distinguished from a process that operates very slowly. It also simplifies the presentation and the proofs.

Next we define the notion of termination. Intuitively, we want to say that a process p_i has terminated in computation x if the process cannot take any further steps. That is, there is no extension of x in which p_i is enabled.

Definition. Process p_i has *terminated* at computation x iff for any extension y of x, $x \overset{i}{\sim} y$.

In this definition a process may terminate without knowing that it has terminated. It is also possible to define termination by using an additional type of event called HALT. Only a process that intends to terminate will perform the halt event. We may then say that p_i has terminated at computation x iff there is a halt event on p_i at x. The results we present in the sequel hold under both definitions. We avoid using HALT in order to simplify the presentation, since its inclusion would force us to add one more property to the definition of an asynchronous protocol.

It is easy to see that for any initial(t) protocol, if process p_i has terminated at computation x then p_i has decided at x. That is, a process has to decide before it terminates. Using the notion of process termination we define what it means for a protocol to terminate. Intuitively, an initial(t) protocol terminates if in any case where no more then t processes initially fail, all correct processes eventually terminate. Before giving the exact definition we extend the notion of fulfilled event (see Section 2), by saying that all events of messages sent to a terminated processes are fulfilled.

Definition. A protocol is *terminating in the presence of t initial* *failures* (abbv. terminating) iff for any set Q of processes where $|Q| \geqslant n-t$, every initial Q-fair sequence is a computation in which any $p_i \in Q$ has terminated.

We say that a problem can be solved without termination [with termination] in the presence of t initial failures iff there exists a [terminating] initial(t) protocol that solves that problem. Obviously every problem that can be solved by a terminating protocol, can also be solved if no requirement is made about termination. However, the opposite claim does not hold. An example proving this fact is the order preserving (renaming) problem, defined next, which can be solved without requiring termination but cannot be solved under the termination requirement for $t \geqslant 2$.

The motivation for the order preserving problem is the need to reduce the size of the name-space of a set of processes. Formally, the order preserving problem is a mapping $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\emptyset\}$, where $|I| >> |D|$, $\bar{I}$ is the set of all vectors with distinct values, and for every $\mathbf{a} \in \bar{I}$ and $\mathbf{d} \in \bar{D}$: $\mathbf{d} \in T(\mathbf{a})$ iff for every i, $j \in \{1,..., n\}$, $a_i < a_j$ implies $d_i < d_j$. The problem was first defined in Ref. 9 where it was proven that the problem is solvable in the crash failure model where up to t processes may fail at any point during an execution, and without requiring termination iff $|D| \geqslant 2^t(n-t+1)-1$. It follows from the theorems presented in the next sections that if termination is not required this result holds also when it is only assumed that the t processes may initially fail, and if termination is required there is no solution (for $t \geqslant 2$) even if $|I| = |D| + 1$ (a proof of that fact appears in Section 7).

Next we prove two technical lemmas that are used later. The lemmas state the relation between input values, output values and initial failures. Lemma 1 says that if some processes take no steps, the remaining ones must still be able to decide. On the other hand, Lemma 2 says that if some processes "wake up late" and begin executing after the others have decided (and terminated), the latecomers must also decide in a manner consistent with the decisions of the other processes. Let $\bar{Q} \equiv N - Q$; as in Ref. 15, let $x[\bar{Q}]y \equiv$ for every $p_i \in \bar{Q}$, $x_i = y_i$. Notice that for an extension y of x, $x[\bar{Q}]\,y$ implies that events on processes that belong to $\bar{Q}$ did not occur in the interval $(y-x)$. Thus $x[\bar{Q}]$ *null* means that no process of $\bar{Q}$ has an event in x. We also use the notation $\langle_{i=1...m}; (y(i)-x)\rangle$ as an abbreviation for $\langle (y(1)-x);...; (y(m)-x)\rangle$, where $y(i)$ is a computation.

Lemma 1. Let $P = (N, C)$ be an asynchronous protocol that solves the problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\emptyset\}$ in the presence of t initial failures (without assuming termination), and let $Q \subseteq N$ be an arbitrary set of processes where $|Q| \geqslant n-t$. For every vector $\mathbf{a} \in \bar{I}$ ($\mathbf{a} \equiv (a_1,..., a_n)$) and computation $x \in C$ where every $p_j \in Q$ either reads the value a_j or does not read any input value and where $x[\bar{Q}]$ *null*, there exists an extension $y \in C$ of x such that $y[\bar{Q}]$ *null*, and any $p_j \in Q$ reads the input value a_j and decides.

Proof. Using $P2$–$P5$, starting from the computation x we can inductively construct an initial Q-fair sequence (w.r.t. P), where every process $p_j \in Q$ reads the input value a_j. Since P can tolerate t initial failures it follows that this sequence has a prefix as required. ∎

Lemma 2. Let $P = (N, C)$ be an asynchronous protocol that solves the problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ in the presence of t initial failure assuming termination, and let $Q \subseteq N$ be an arbitrary set of processes where $|Q| = n - t$. Let $\mathbf{a} \equiv (a_1,..., a_n)$ be an arbitrary vector from $\bar{I}$ and let $x \in C$ be a computation where $x[\bar{Q}]$ *null* and every $p_j \in Q$ reads the value a_j, decides on some value d_j, and terminates. For any $p_i \notin Q$ there exists a computation $y(i) \in \mathit{Extension}(x, i)$, where p_i reads the input value a_i and decides on some value d_i such that $\mathbf{d} \in T(\mathbf{a})$, $(\mathbf{d} \equiv (d_1,..., d_n))$.

Proof. As in the previous proof, using $P2$–$P5$, starting from the computation x we can inductively construct for each $p_i \notin Q$ an initial $(Q \cup \{p_i\})$-fair sequence (w.r.t. P) where process p_i reads the input value a_i. Since P can tolerate t initial failures it follows that this sequence has some prefix $y(i)$ in which p_i read the input value a_i and decides on some value, say d_i. It follows from $P1$, $P2$ and $P3$ that $P3$ that $\langle x;_{p_i \notin Q}; (y(i) - x)\rangle$ is a computation, and hence $\mathbf{d} \in T(\mathbf{a})$. ∎

In the next sections we examine the possibility and impossibility of solving problems in an asynchronous environment where t processes may initially fail. Initially, we do not require a protocol (that solves a problem) to terminate and then subsequently, we add the termination requirement and examine how it affects the results.

5. SOLVABILITY WITHOUT TERMINATION

In this section we characterize the problems that can be solved in an asynchronous environment where $0 \leqslant t < n/2$ processes may initially fail. We use the intuitive appeal of a game-theoretical characterization by reducing the question of solvability in the model under consideration to whether there is a winning strategy to a particular game described next. The exposition here is influenced by the "Ehrenfeucht Game,"(17) which is used in mathematical logic to determine if two structures are elementarily equivalent. (That is, if they satisfy the same first-order sentences.) A similar result holds for an asynchronous shared memory model.(13)

The game $G_1(T, t)$, corresponding to a problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ and a natural number t, is played by two players A (Adversary) and B, according to the following rules. Each play of the game begins with a move of player A and in the subsequent moves both players move alternately. The game is played on a board with n empty circles drawn in a straight line. The circles are numered from 1 to n. At the first move player A chooses $n-t$ input values from (the set of input values) I and "places" them on arbitrary $n-t$ empty circles. Then player B chooses $n-t$ decision values from (the set of decision values) D and uses them to *cover* all the $n-t$ input values placed by player A in the previous move. Subsequent moves consist of player A choosing a *single* value from I in each move, and placing it on an empty circle, and then player B choosing a single value from D and covering the previous value placed by player A. The play is completed when all the n circles are covered with decision values from D. We emphasize that at any time each player knows all the previous moves.

We denote by $a_i \in I$ and $d_i \in D$ the values players A and B, respectively, placed on the ith circle in the course of the play. For simplicity we assume that the final vector $(a_1, ..., a_n)$ belongs to $\bar{I}$. Player B has *won* the play iff $\mathbf{d} \in T(\mathbf{a})$. Player B has a *winning strategy* in the game $G_1(T, t)$, denoted B *wins* $G_1(T, t)$, if B can always win each play.

We also assume that the processes have distinct identities that are mutually known. We will remove those assumptions at the end of the section.

Theorem 1. A problem T can be solved, without assuming termination, in a completely asynchronous environment where t processes may initially fail iff player B wins $G_1(T, t)$.

Proof. We first prove the **if** direction. The proof is based on the fact that in the model under consideration, it is possible to elect a leader.[2] Suppose player B has a winning strategy in the game $G_1(T, t)$. We describe a protocol that solves T in the presence of t initial failures. The protocol starts by first electing one of the processes as a leader. After the leader broadcasts its identity, every correct process sends its own input value and its id to the leader. Since at most t processes might be faulty, the leader is gauranteed to receive $n-t$ input values (including its own). Then the leader examines the winning strategy of player B to determine the corresponding $n-t$ decision values, and sends the relevant decision value to each process from which it received an input value. Afterwards the leader just waits. Upon receiving an additional input value, by using the winning strategy of player B, it produces the right decision value, and sends it to the process from which it received the input value. Each process that receives a decision value from the leader decides on that value.

We now prove the **only if** direction. Let $P = (N, C)$ be an asynchronous protocol that solves T in the presence of t initial failures. We describe a winning strategy for player B in $G_1(T, t)$. Let $\mathbf{a} \in \bar{I}$ be an arbitrary input vector, and let Q be a set of processes where $|Q| = n - t$ such that player A chooses in his first move the set of values $\{a_i | p_i \in Q\}$ and places each value a_i on the circle numbered with i. From Lemma 1, there exists a computation $x \in C$ such that $x[\bar{Q}]$ *null*, any process $p_i \in Q$ reads the input value a_i and decides in x. Let d_i denote the value on which process $p_i \in Q$ decided in x. By using P, player B can simulate the computation x, output the $n - t$ decision values, and cover each input value a_i by the corresponding decision value d_i. Assume that player A next chooses some value a_j where $p_j \notin Q$ and places it on the jth circle. By Lemma 1, there is an extension y of x in which process p_j reads the input value a_j and decides on some value d_j, and where $x[\bar{Q} - \{p_j\}]y$. Thus again, by using P, player B can continue the simulation of x in order to simulate computation y and choose d_j. A similàr construction holds also for any further input values that A chooses. Finally, since P solves T, $\mathbf{d} \in T(\mathbf{a})$ and hence player B wins the game. ▌

Examples of problems that can be shown to be unsolvable using this theorem (in the model under consideration), are transaction commitment, sorting and rotating(t). To show the impossibility for transaction commitment we demonstrate that B has no winning strategy. The adversary can choose as its first move $n - t$ "1" values; B then must also use $n - t$ "1" values since player A may later choose only "1" values. Then A can add the value "0" and B loses. This theorem also points out how to construct a solution (*i.e.*, a protocol) for any solvable problem T. First find a winning strategy for player B in the game $G_1(T, t)$ and then plug it into the (schematic) protocol presented in the "if" part of the proof of Theorem 1.

Corollary 1. The message complexity of electing a leader is an upper bound for solving any other problem on a complete network in a completely asynchronous environment where t processes may initially fail.

Proof. The (schematic) protocol presented in the "if" part of the proof of theorem 1, which can solve any solvable problem in this model, first elects a leader and then uses at most $2n$ additional messages. Since $\Omega(n)$ is clearly a lower bound for electing a leader, the result is proven. ▌

We mention that the message complexity for electing a leader in *reliable* complete networks is $\theta(n \log k)$, where k is the number of processes that start the protocol.[18] For complete networks where t processes may initially fail the message complexity for electing a leader is $\theta(n \log k + kt)$.[2]

The last corollary can be generalized to an arbitrary network by considering the number of messages needed to collect the input values and then return the final decision values from the leader.

Remark. In order to prove that for a given network there is a *fixed* upper bound on the message complexity (that depends only on the network size) for solving any problem in the initial failure model, we have strongly used the fact that it is possible to elect a leader in that model. It is interesting to note that for the (slightly stronger) crash failure model in which it is not possible to elect a leader (see Ref. 6), it has been proven that no such fixed bound exists.(8) More precisely, there exist problems such that any protocol solving them in the presence of crash failures must send arbitrarily many messages, for some scenarios.

In the proof of Theorem 1, the assumption that the processes have distinct identities that are mutually known is used at the point where the leader (in the "if" part) has to consult with the winning strategy of player B. We now remove this assumption and only require the input values are disrinct. (This, of course, also covers the case where the processes have distinct identities which are not mutually known). Next, we modify Theorem 1 so that it holds under this weaker requirement.

Recall that $\mathbf{a}$ and $\mathbf{d}$ are the vectors players A and B, respectively, placed in the course of the play. Let $\pi \equiv (\pi_1, ..., \pi_n)$ be a permutation of $\{1, ..., n\}$, and let $\pi(\mathbf{a})$ denote the vector $(a_{\pi_1}, ..., a_{\pi_n})$. We say that player B *strongly won* the play iff for every permutation π of $\{1, ..., n\}$ where $\pi(\mathbf{a}) \in \bar{I}$ it is the case that $\pi(\mathbf{d}) \in T(\pi(\mathbf{a}))$. Player B has a *strong winning strategy* in the game $G_1(T, t)$ if it is possible for him to strongly win each play. (This is denoted by "*B strongly wins $G_1(T, t)$*".) If we now substitute in Theorem 1 the term "strongly wins" for "wins" then the modified theorem will hold under the requirement that the input values are distinct, and with no need to assume anything about the process identities. The proof of this theorem involves some technical modification of the previous proof, and is based on the fact that a leader can stil be elected. Another way of resolving this problem is the following. We say that a problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\emptyset\}$ is *symmetric* iff for every vector $\mathbf{a} \in \bar{I}$ and for every permutation π of $\{1, ..., n\}$, it is the case that $\pi(\mathbf{a}) \in \bar{I}$ and $\pi(\mathbf{d}) \in T(\pi(\mathbf{a}))$. For symmetric problems the notion of *strongly wins* and *wins* coincide, and hence for such problems the original formulation of Theorem 1 still holds (without the assumption that the processes have distinct identities). We mention that the notion of strongly wins makes sense only when dealing with symmetric problems.

6. SOLVABILITY WITH TERMINATION

In this section we add the termination requirement and, as in the presentation of the previous section, we characterize the problems that can be solved by a terminating protocol in an asynchronous environment where $0 \leqslant t < n/2$ processes may initially fail. With this requirement it is still possible to elect a leader, however now the leader cannot coordinate all activities and decisions as before, since it itself is also required to terminate. Put another way, a process may start operating long after the leader has terminated. We note that if it is only required that all correct processes *except one* eventually terminate then the result from the previous section (*i.e.*, Theorem 1) holds.

In the sequel we modify the game $G_1(T, t)$ in order to reflect termination. In the new game the role of player A is left unchanged while the *power* of B is restricted: each time B makes a move it "forgets" all previous moves except the first move of both players.

The new game $G_2(T, t)$, corresponding to a problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ and a natural number t, is played by player A (Adversary) and a group of players $B \equiv \{B_1, ..., B_{t+1}\}$ according to the following rules. Each play of the game begins with a move of player A and in the subsequent moves both player A and some player from B move alternately. Each player $B_i \in B$ takes exactly one move in each play. The game is played (as before) on a board with n empty circles (numbered from 1 on n) drawn in a straight line. As his first move player A chooses $n-t$ input values from I and "places" them on arbitrary $n-t$ empty circles. Then player B_1 chooses $n-t$ decision values from D and uses them to *cover* all the $n-t$ input values placed by player A in the previous move. The subsequent moves consist of player A choosing in its ith move a *single* value from I and placing it on an empty circle and then player B_i choosing, based *only* on the values placed by player A in the first and current moves and the values placed by player B_1, a single value from D and covering the last value placed by player A. We emphasize that in this game player B_i does not know all the previous moves. He knows only the first moves of players A and B_1 and the ith move of player A. (This can be implemented by hiding from a player the circles he is not supposed to see.) The play is completed when all the n circles are covered with decision values from D.

We denote by $a_i \in I$ and $d_i \in D$, respectively, the values that player A and some player in B placed on the ith circle during the play. For simplicity we assume that the final vector $(a_1, ..., a_n)$ belongs to $\bar{I}$. As previously, the group of players B has *won* the play iff $\mathbf{d} \in T(\mathbf{a})$. The group of players B has a *winning strategy* in the game $G_2(T, t)$, denoted B *wins* $G_2(T, t)$, if it can always win each play. As before, we have assumed that the processes have distinct identities that are mutually known. Removing this assumption is similar to the treatment in the previous section.

Theorem 2. A problem T can be solved, assuming termination, in a completely asynchronous environment where t processes may initially fail iff B wins $G_2(T, t)$.

Proof. We first prove the **if** direction. Suppose B has a winning strategy in the game $G_2(T, t)$. We describe a terminating protocol that solves T in the presence of t initial failures. The protocol starts by first electing one of the processes as a leader. After the leader broadcasts its identity, every correct process sends its own input value and id to the leader. Since at most i processes might be faulty, the leader is guaranteed to receive $n-t$ input values (including its own). Then the leader consults with the winning strategy of B (by playing the role of B_1) to determine the corresponding $n-t$ decision values. After that the leader broadcasts to all processes a message which contains $n-t$ triples. Each triple consists of a process' id and the input and decision values of that process. The leader then decides and terminates. Each process p_i that receives the above message from the leader first checks if its id appears in one of the triples. If it does, then the process decides on the decision value that appears in that triple and terminates. Otherwise, using the message from the leader and its own input, p_i consults with the winning strategy of B, by playing the role of one of the players in B, decides and terminates.

We now prove the **only if** direction. Let $P=(N, C)$ be a terminating asynchronous protocol that solves T in the presence of t initial failures. We describe a winning strategy for B in $G_2(T, t)$. Let $\mathbf{a} \in \bar{I}$ be an arbitrary input vector, and let Q be a set of process where $|Q|=n-t$ such that player A chooses in his first move the set of values $\{a_i | p_i \in Q\}$ and places each value a_i on the circle numbered with i. From Lemma 1 and the termination assumption, there exists a computation $x \in C$ such that $x[\bar{Q}]null$, any process $p_i \in Q$ reads the input value a_i, decides, and terminates in x. Let d_i denote the value decided upon by process $p_i \in Q$ in computation x. By using P, player B_1 can simulate the computation x, output the $n-t$ decision values, and cover each input value a_i by the corresponding decision value d_i. Now assume that player A chooses in its kth move some value a_j where $p_j \notin Q$ and places it on the jth circle. By Lemma 2, (for any j) there is a computation $y(j) \in Extension(x, j)$ in which process p_j reads the input value a_j and decides on some value d_j. By using P, player B_k can continue the simulation of x in order to simulate computation $y(j)$ and choose d_j. A similar construction holds also for any input values that A chooses. From Lemma 2, $\mathbf{d} \in T(\mathbf{a})$ and hence B wins the game. ▌

As already mentioned, it is easy to show by using Theorem 2 that (due to reasons of symmetry) the order preserving problem cannot be solved assuming termination for $t \geqslant 2$. (A proof is given in the next section.) Examples of solvable problems (for $t < n/2$) are consensus and leader election. We note that Corollary 1 (from the previous section) also holds in this model.

Remark. In the initial failure model there are nontrivial problems that can be solved assuming termination. This is not the case in another model that also has what might be considered a weak type of failures. It is proved by Koo and Toueg,(19) that no nontrivial protocol can be guaranteed to terminate in a model where only *transient* communication failures can occur. (Channel failures are transient if any message sent repeatedly is eventually received.)

7. A TWO PLAYER GAME WHICH REFLECTS TERMINATION

In the previous section we presented a game with $t+2$ players and showed how to use the game in order to decide if a problem can be solved when termination is required. In this section we describe a game with only two players for the same purpose. The new game is less general than the previous one in the sense that it is useful only when applied to problems where the set of possible input values is at most countable.

The game $G_3(T, t)$, corresponding to a problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ and a natural number t, is played by two players A (Adversary) and B, according to the following rules. Each play of the game begins with a move of player A and in the subsequent moves both players move alternately. The game is played (as before) on a board with n empty circles drawn in a straight line. The circles are numbered from 1 to n. At the first move player A chooses $n-t$ input values from I and "places" them on arbitrary $n-t$ empty circles. Then player B chooses $n-t$ decision values from D and uses them to *cover* all the $n-t$ input values placed by player A in the previous move. The subsequent moves consist of player A choosing a *single* value from I and placing it on a (possibly nonempty) circle that is not one of the $n-t$ circles it used in the first move, and then player B choosing a single value from D and covering the previous value placed by player A. The fact that nonempty circles can be reused is equivalent to A "changing his mind" about some of the later moves. The play is completed upon the decision of A, where the last move is always made by B. (The result holds also in a variation of the game in which player A announces the number of moves before the play begins.)

We denote by $(a_i, d_i) \in I \times D$ some pair of values that players A and B

placed on the ith circle in the course of the play in two successive moves (i.e., B covered the value a_i with d_i). For simplicity we assume that at any moment the last value A placed may be completed together with the $n-t$ values placed by player A in its first move to a vector in $\bar{I}$. Player B has *won* the play iff for *every* possible n-tuple of pairs $((a_1, d_1),..., (a_n, d_n))$ chosen from pairs placed in the course of the play, $\mathbf{d} \in T(\mathbf{a})$ or $\mathbf{a} \notin \bar{I}$. Player B has a *winning strategy* in the game $G_3(T, t)$, denoted B *wins* $G_3(T, t)$, if it can always win each play no matter what A does.

As before, we assume that the processes have distinct identities that are mutually known. Removing this assumption is similar to the treatment in the previous sections.

Theorem 3. A problem $T: \bar{I} \rightarrow 2^{\bar{D}} - \{\varnothing\}$ where I is at most countable can be solved assuming termination in a completely asynchronous environment where t processes may initially fail iff player B wins $G_3(T, t)$.

Proof. We first prove the (more difficult) **if** direction. Suppose B has a winning strategy in the game $G_3(T, t)$. We describe a terminating protocol that solves T in the presence of up to t initial failures. The protocol starts by first electing one of the processes as a leader. Then every correct process sends its own input value and id to the leader. Since at most t processes might be faulty, the leader is guaranteed to receive $n-t$ input values (including its own). Then the leader consults with the winning strategy of B to determine the corresponding $n-t$ decision values. After that the leader broadcasts to all processes a message containing $n-t$ triples. Each triple consists of a process' id and the input and decision values of that process. The leader then decides and terminates. Each process p_i that receives this message from the leader first checks if its id appears in one of the triples. If it does, then the process decides on the decision value that appears in that triple and terminates. Otherwise, using the message from the leader and its own input, p_i exploits the winning strategy of B (in a manner described next), decides, and terminates.

The process p_i will simulate a particular play of the game in order to determine the value upon which it will decide. We assume that the set of input values I is infinite. The treatment when I is finite is similar (and simpler). Since I is countable we assume w.l.o.g. that I is the set of natural numbers. (There is a bijection α from the natural numbers into I, so I can be represented as $\{\alpha(m) \mid m$ is a natural number$\}$.) We say that player A uses the *order strategy* if after A's first move, in all its subsequent moves A covers the circles from left to right (cyclicly) skipping the $n-t$ circles it covered in its first move. Moreover, each time A has to place a number on a circle it always chooses the smallest possible number which is greater than the previous number A placed on that circle. (A number is *possible* if, together with the $n-t$ values placed by player A in its first move, it can be completed to a vector in $\bar{I}$.) We now show how process p_i uses the winning strategy of B to simulate a play of the game. Using the message from the leader it simulates the first moves of A and B by placing each of the $n-t$ input and output values on the corresponding circles. Then it continues assuming that A plays according to the order strategy, and it uses B's winning strategy to simulate the moves of B. Process p_i stops the simulation one move after A places a value identical to p_i's own input value on the ith circle, and p_i decides on the value that B uses to cover that input value. Since A plays at the simulation according to the order strategy, p_i is sure that eventually A will place its input value on the ith circle. Since p_i decides on a value according to B's winning strategy, this value must be consistent with any possible decision value of the other processes chosen in the same way.

We now prove the **only if** direction. Let $P=(N, C)$ be a terminating asynchronous protocol that solves T in the presence of t initial failures. We describe a winning strategy for B in $G_3(T, t)$. Let $\mathbf{a} \in \bar{I}$ be an arbitrary input vector, and let Q be a set of processes where $|Q| = n-t$, such that player A chooses in his first move the set of values $\{a_i | p_i \in Q\}$ and places each value a_i on the circle numbered with i. From Lemma 1 and the termination assumption, there exists a computation $x \in C$ such that $x[\bar{Q}]$*null*, any process $p_i \in Q$ reads the input value a_i, decides, and terminates in x. Let d_i denote the value decided upon by process $p_i \in Q$ in computation x. By using P, player B can simulate the computation x, output the $n-t$ decision values, and cover each input value a_i by the corresponding decision value d_i. Now assume that player A chooses in one of its moves the value a_j where $p_j \notin Q$ and places it on the jth circle. By Lemma 2, there is a computation $y(j) \in \mathit{Extension}(x, j)$ in which process p_j reads the input value a_j and decides on some value d_j. By using P, player B can continue the simulation of x in order to simulate computation $y(j)$ and choose d_j. Note that the value d_j is based *only* on the values placed by players A and B in the first moves and the value placed by player A in the current move. A similar construction holds for any input value that A chooses. From Lemma 2, $\mathbf{d} \in T(\mathbf{a})$ and hence B wins the game. (In this part of the proof we do not use the fact that the cardinality of I is at most countable.) ▌

The requirement that the set of input values is at most countable is necessary for the correctness of Theorem 3. Consider the following problem: each process starts with some input value that is a *real* number, and it is required that at least one process outputs a *finite* set of values containing the input values of all other processes. In the Appendix, we show that there is no terminating protocol that solves this problem in the presence of two or more initial failures. On the other hand, player B has the following winning strategy for the game corresponding to this problem: each time B has to make a move it places a set containing all input values that A has already placed anywhere on the board. Thus the existence of a winning strategy in the game does not imply the existence of a terminating protocol when the input domain is not countable.

Note that if we replace the assumption that the input values are the real numbers by the assumption that they are the rationals (and hence countable), then since B has a winning strategy (just as before) it follows from Theorem 3 that there exists a protocol that solves the problem. As one example of a solution, a mapping of the input values onto the natural numbers can be fixed, and each process could output the finite set of all the rationals with a mapping less than or equal to that of its input value.

We now prove that the order preserving problem cannot be solved assuming termination for $t \geqslant 2$. Player A uses the following strategy: In the first move it chooses the $n-2$ smallest values. Then, in all its subsequent moves A covers the other two circles alternately; each time, A chooses a value greater than all values it had chosen previously. Player A stops the game after $|D|+1$ move ($|D|$ is the cardinality of the set of output values). It is not hard to show that player B must choose $|D|+1$ distinct values, which is clearly impossible. Since the game has no winning strategy, by Theorem 3 there is no protocol to solve it in the presence of initial failures, assuming termination.

8. DISCUSSION

In this paper we introduced three simple games and reduced the question of whether a certain problem can be solved in completely asynchronous systems where a number of processes may fail prior to the execution to the question of whether there is a winnning strategy for one of these games. We gave several examples demonstrating that it is very convenient to reason about problems using such games. The differences which rose in the proposed games when termination is not required and when it is necessary help to clarify the price of this added requirement.

An upper bound on the message complexity for solving any problem in the model was proved, by showing that the message complexity of any solvable problem is bounded by the message complexity for solving the leader election problem. We showed how to use a given solution for the election problem in order to construct a solution to any other solvable problem. This implies that in some sense we may think of the election problem as a "complete" problem for the initial failure model.

It follows from our results together with the results in Ref. 12 that in a message passing system where termination is required, a problem can be solved by a *deterministic* protocol that can tolerate up to t *initial* failures if and only if the problem can be solved by a *randomized* protocol that can tolerate up to t *crash* failures. This result also holds for the asynchronous shared memory model (without assuming termination).(13)

It follows from our results that for initial failures, there exists a strict resiliency hierarchy. That is, for each $0 < t < n/2$ there are problems that can be solved in the presence of $t-1$ failures but cannot be solved in the presence of t failures.

Following our results here, a result similar to Theorem 1, with a similar proof, was shown to hold for an asynchronous shared memory model that supports only atomic read and write operations [see Ref. 13, Section 7]. A similar result holds for other models where it is possible to elect a leader in the presence of initial failures, for example, a synchronous model in which correct processes may join the computation at any round.

APPENDIX

We present an example demonstrating that the requirement that the set of input values is at most countable is necessary for the correctness of Theorem 3. Consider the following problem, T_f. Each process starts with some input value, and it is required that at least one process outputs a *finite* set of values containing the input values of all processes.

Lemma A1. Let D be the set of all possible input values. Then for any $N \geqslant 5$, there is a terminating protocol that solves T_f in the presence of two initial failures iff there exists a function f from D to finite subsets of D such that for every pair of values u and v, $u \in f(v)$ or $v \in f(u)$.

Proof. **If** Let $N = k+2$, and assume that there is such a protocol. For brevity, assume that D contains the integers, and consider a computation x of this protocol in which processes $p_1,..., p_k$ have inputs $1,..., k$ resp., and the remaining two processes are asleep until these processes decide and terminate. Let S_i be the decided upon by process p_i.

Now continue the computation x by having p_{k+1} wake up with input u and run until it decides on a set S_u. Similarly, continue the earlier computation x (without p_{k+1}) by having p_{k+2} wake up with the same input u and run until it decides on a set T_u. $f(u)$ is defined by: $f(u) = \bigcup_{i=1}^{k} S_i \cup S_u \cup T_u$. By Lemma 2 and by the definition of the problem, we must have that for every pair u and v, $(S_1,..., S_k, S_u, T_v)$ is a legal output vector. This means that for every such pair, $u \in f(v)$ or $v \in f(u)$, since otherwise no set in the output vector contains both u and v.

Only if. Assume that f is such a function. It is easily observed that the problem can be solved by the following terminating protocol: First, a leader is elected, and this leader decides on the set S of all inputs known to it (there are at least $n-2$ such inputs). The leader then broadcasts this set and terminates. Every nonleader process with input u waits until it receives the set S, and it decides on $f(u) \cup S$. ▌

We now show that the condition of this *above* Lemma does not hold for $D = \mathbf{R}$, the set of real numbers. This means that no terminating protocol for solving T_f exists for this case.

Theorem A1. Let $\mathbf{R}$ be the set of real numbers. Then there is no function f from $\mathbf{R}$ to finite subsets of $\mathbf{R}$ such that each u and v in $\mathbf{R}$, either $f(u)$ contains v or $f(v)$ contains u.

Proof [Ref. 20]. Assume to the contary that there is such a function f. Let $\mathbf{N} \subseteq \mathbf{R}$ be the set of integers, and let $D = \bigcup_{i \in N} f(i)$. Then D is countable. Since $\mathbf{R}$ is uncountable, there is an $r \in \mathbf{R}$ which is not in D. Thus, r is not in $f(i)$ for each integer i. Hence, $f(r) \supseteq \mathbf{N}$. But this is impossible, since N is not finite. ▌

Note. If we assume that the function f is from the reals into the countable subsets of the reals (rather than into finite subsets) such that for every pair of reals u and v, $u \in f(v)$ or $v \in f(u)$, and we assume the axiom of choice, then the existence of such a function is equivalent to the Continuum Hypothesis. Hence, neither the existence nor the nonexistence of such a function is derivable from the standard axioms of set theory with the axiom of choice.(17)

REFERENCES

1. M. Fischer, N. Lynch, and M. Paterson, Impossibility of distributed consensus with one faulty process, *JACM* **32**(2):374–382 (1985).
2. R. Bar-Yehuda, S. Kutten, Y. Wolfstahl, and S. Zaks, Making distributed spanning tree algorithms fault-resilient, *STACS 87*, *LNCS* No. 247, pp. 432–445.
3. H. Abu-Amara, Fault-Tolerant distributed algorithm for election in computer network, *IEEE Transactions on Computers*, **37**(4):449–453 (April 1988).
4. D. Dolev, C. Dwork, and L. Stockmeyer, On the minimal synchronism needed for distributed consensus, *JACM* **34**(1):77–97 (1987).
5. C. Dwork, N. Lynch, and L. Stockmeyer, Consensus in the presence of partial synchrony, *JACM* **35**(2):288–323 (1988).
6. S. Moran and Y. Wolfstahl, An extended impossibility result for asynchronous complete networks, *IPL* **26**:145–151 (November 1987).

7. G. Taubenfeld, Impossibility Results for Decision Protocols, Technion Technical Report #445 (January 1987). Revised version, Technion Technical Report #506 (April 1988).
8. O. Biran, S. Moran, and S. Zaks, A Combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor, *ACM-PODC* (1988).
9. H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, and R. Reischuk, Achievable cases in an asynchronous environment, *IEEE-FOCS*, pp. 337–346 (1987).
10. M. Bridgland and R. Watro, Fault-tolerant decision making in totally asynchronous distributed systems, *ACM-PODC*, pp. 52–63 (1987).
11. G. Taubenfeld, S. Katz, and S. Moran, Impossibility Results in the presence of multiple faulty processes, Technion Technical Report #492 (January 1988). Also, in the *Proceeding of the 9th FCT-TCS Conference*, Bangalore, India (December 1989).
12. B. Chor and L. Moscovici, Solvability in asynchronous environments, *IEEE-FOCS*, pp. 422–427 (1989).
13. G. Taubenfeld and S. Moran, Possibility and impossibility results in a shared memory environment, *Proceedings of the 3rd International Workshop on Distributed Algorithms*, Nice, France (September 1989). In: *LNCS* 392, eds., J. C. Bermond and M. Raynal, Springer Verlag (1989).
14. M. Chandy and J. Misra, On the nonexistence of robust commit protocols, Unpublished manuscript (November 1985).
15. M. Chandy and J. Misra, How processes learn, *Distributed Computing*, pp. 40–52 (1986).
16. V. Hadzilacos, A knowledge theoretic analysis of atomic commitment protocols, *ACM-PODS*, pp. 129–134 (1987).
17. H. Ebbinghaus, J. Flum, and W. Thomas, Mathematical Logic, Springer-Verlag (1984).
18. E. Korach, S. Moran, and S. Zaks, Tight lower and upper bounds for some distributed algorithms for a complete network of processors, *ACM-PODC*, pp. 199–207 (1984).
19. R. Koo and S. Toueg, Effects of message loss on the termination of distributed protocols, *IPL* **27**:181–188 (April 1988).
20. G. Moran, personal communication.