A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch [*]
Lab for Computer Science
MIT, Cambridge, MA 02139
lynch@tds.lcs.mit.edu

1 Introduction

This talk is about impossibility results in the area of distributed computing. In this category, I include not just results that say that a particular task cannot be accomplished, but also lower bound results, which say that a task cannot be accomplished within a certain bound on cost.

I started out with a simple plan for preparing this talk: I would spend a couple of weeks reading all the impossibility proofs in our field, and would categorize them according to the ideas used. Then I would make wise and general observations, and try to predict where the future of this area is headed. That turned out to be a bit too ambitious; there are many more such results than I thought. Although it is often hard to say what constitutes a "different result", I managed to count over 100 such impossibility proofs! And my search wasn't even very systematic or exhaustive.

It's not quite as hopeless to understand this area as it might seem from the number of papers. Although there are 100 different results, there aren't 100 different ideas. I thought I could contribute something by identifying some of the commonality among the different results.

So what I will do in this talk will be an incomplete version of what I originally intended. I will give you a tour of the impossibility results that I was able to collect. I apologize for not being comprehensive, and in particular for placing perhaps undue emphasis on results I have been involved in (but those are the ones I know best!). I will describe the techniques used, as well as giving some historical perspective. I'll intersperse this with my opinions and observations, and I'll try to collect what I consider to be the most important of these at the end. Then I'll make some suggestions for future work.

*This work was supported in part by the National Science Foundation (NSF) under Grant CCR-86-11442, by the Office of Naval Research (ONR) under Contract N00014-85-K-0168 and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125.

2 The Results

I classified the impossibility results I found into the following categories: shared memory resource allocation, distributed consensus, shared registers, computing in rings and other networks, communication protocols, and miscellaneous.

2.1 Shared Memory Resource Allocation

This was the area that introduced me not only to the possibility of doing impossibility proofs for distributed computing, but to the entire distributed computing research area.

In 1976, when I was at the University of Southern California, Armin Cremers and Tom Hibbard were playing with the problem of *mutual exclusion* (or allocation of one resource) in a shared-memory environment. In the environment they were considering, a group of asynchronous processes communicate via shared memory, using operations such as read and write or test-and-set.

The previous work in this area had consisted of a series of papers by Dijkstra [38] and others, each presenting a new algorithm guaranteeing mutual exclusion, along with some other properties such as progress and fairness. The properties were specified somewhat loosely; there was no formal model used for describing algorithms and specifying problems to be solved. Each paper, in fact, seemed to solve a slightly different problem (involving different fairness, performance and fault-tolerance properties). It was difficult to compare the results in the different papers.

Cremers and Hibbard thought about inherent limitations on the solvability of mutual exclusion in that environment, for the special case where memory was accessible via powerful test-and-set primitives. (Their version of test-and-set was very general, allowing one atomic access to shared memory to read, compute and write a value back.) An obvious complexity measure to study was the size of shared memory; they considered the very simple problem of achieving mutual exclusion between two processes, using a single shared variable, and asked how many values the shared variable would need to take on. A 2-valued semaphore is plenty if there are no fairness requirements; however, if fairness is included then 3 values were the best they could do. They proved the simple result that 2 values were insufficient.

In order to do this, they had to embark on a major modeling effort. (To see how important the modeling work was here, note that the title of their paper [35] emphasizes their model rather than their combinatorial result.) The algorithms work had proceeded quite far without anyone having defined a formal model or being too precise about problem statements. But in order to give a formal proof of even a very simple impossibility result, Cremers and Hibbard needed the rigor of a formal model. This model needed to have two separate components - a careful description of the correctness conditions (mutual exclusion, progress and fairness), and a careful description of the space of allowable implementations, i.e., processes and shared memory.

Defining the model was hard work, especially the problem statement. The mutual exclusion condition was easy to define, but the progress and fairness conditions were not. For instance, the requirements involved the system "continuing to make progress". But clearly no system could guarantee progress if the processes were permitted to stop at arbitrary times during their protocols. They needed a notion of *admissible execution* that described exactly when processes were required to continue taking steps (e.g., while engaging in a protocol to obtain a resource, but not necessarily at other times).

They also needed to capture some ideas about *who controls each action*. For example, they needed to capture the idea that each process "might request the resource at any time", i.e., that the requesting actions were not under the control of the mutual exclusion algorithm. Otherwise, they would risk having a trivial problem statement that permits the solution algorithm to prevent processes from making requests.

They also needed to express conditional statements like "the system is required to guarantee progress *if the environment cooperates in that progress*" - e.g., the system will repeatedly grant the resource provided the environment always returns it. They ended up with a carefully-crafted and delicate set of axioms for their problem statement.

They proved their impossibility result for 2 values by assuming that memory was 2-valued, and carrying out a proof by contradiction using a case analysis. This involved constructing several finite runs of the algorithm, in which the processes request the resource and take steps in various orders. Consider the values that the memory takes on at the end of all of these runs. Since there are only two values, the pigeonhole principle implies that there are many situations in which the memory must have the same value. They showed by case analysis that no matter how values get assigned, there must be two "incompatible" situations in which shared memory has the same value, and in which one of the processes also has the same state, even though these two situations require different behavior from the process in order to satisfy the correctness conditions. For example, suppose that shared memory could have the same value and process p_1 have the same state, in two situations - one where p_2 is in its critical region and one where it is not requesting anything. In the second situation, p_1 must eventually go to its critical region on its own, whereas in the first, that would violate mutual exclusion. These two situations are indistinguishable to p_1, and so it must behave in the same way in both situations. But then one or the other situation would lead to incorrect behavior, a contradiction.

This simple result already demonstrated the basic idea upon which all the 100 impossibility proofs in distributed computing are based - *the limitations imposed by local knowledge*. (A process in this shared memory architecture could be said to "know" only what is in its local state and in the shared memory, since that is all that it can see directly.) It also demonstrated the importance of formal models for stating and proving impossibility results.

This early work influenced two different kinds of later work: that on mutual exclusion upper and lower bounds, and that on models for distributed computing.

A couple of years later, at Georgia Tech, I began working in distributed computing, mainly because there was a lot of activity there on design of distributed systems. With Mike Fischer and Jim Burns, I began trying to understand what the interesting theoretical ideas were in this new research area. One of the first things we did was to go back and look at the mutual exclusion work, in particular, that of Cremers and Hibbard.

In [26], we extended the results of [35] to n processes rather than 2 (but still considered just one shared variable). The extended results turned out to be very sensitive to assumptions about fairness:

1. Any solution that exhibits *bounded waiting*, where there is a bound on how many times any process can bypass any other while the latter is waiting, requires at least $n+1$ values.

2. Even if no such bound is required, if *no lockout* is required, then $\Omega(\sqrt{n})$ values are required.

3. Adding a technical assumption to the preceding case, that processes cannot remember what they did on previous times through the protocol, raises the lower bound to $n/2$. (It is an open question whether this technical assumption is necessary.)

The arguments are basically similar to those of [35], based on the pigeonhole principle applied to values of shared memory, only in place of case analysis there is a more systematic examination of executions.

The first result uses a version of the following idea. Suppose p_1 enters the system and goes to its critical region. Then $p_2, \cdots, p_n$ enter the system in turn, each stopping at a point where it is waiting for a chance to enter its critical region. Consider all the values of the variables immediately after the steps of $p_2, \cdots, p_n$. If any p_i and p_j leave the variable with the same value, $i < j$, then the situation C_j in which $p_1, \cdots, p_j$ all enter "looks like" the situation C_i in which only $p_1, \cdots, p_i$ enter, to $p_1, \cdots, p_i$. Starting from situation C_i, $p_1, \cdots, p_i$ are able on their own to enter and leave the critical region arbitrarily many times; therefore, they are also able to do this starting from C_j. But this means that in situation C_j they can bypass a stopped p_j arbitrarily many times, more times than allowed by the bound for bounded waiting. This is a contradiction.

This proof doesn't work if the fairness assumption is weakened to allow unbounded bypassing but no lockout. A violation of bounded waiting occurs in finite time, so in showing that such a violation occurs it suffices to construct a finite bad execution. A demonstration of lockout, however, requires an infinite admissible execution in which some process gets locked out. We can't modify the construction above to permit $p_1, \cdots, p_i$ to bypass p_j infinitely often, while p_j just sits there, because p_j is required to take steps every so often. In the situation above, as soon as p_j takes its next step, it might reveal its presence to $p_1, \cdots, p_i$, so they no longer have the requisite limited knowledge.

The lower bounds for no lockout use trickier constructions. The contradictions involve the construction of incompatible infinite admissible executions that look the same to particular processes, who get fooled thereby and exhibit incorrect behavior. The proper treatment of admissibility was one of the most difficult aspects of this work.

This work is a good example of the interesting "game" of working on conflicting positive and negative results at the same time. We were working on trying to raise the lower bound of $n/2$ for no lockout algorithms to n, since it seemed very unlikely that n processes could arbitrate among themselves fairly if there weren't even enough values of shared memory for all the processes to uniquely record their presence. But that intuition turned out to be false – we came up with a complicated algorithm that used only around $n/2$ values! The algorithm arose in the course of trying to prove impossibility – carefully examining the reasons why all the plausible ideas for impossibility proofs failed suggested what features a correct algorithm would have to have - and then one with these features actually worked. This algorithm was not practical; rather, it was a kind of algorithm I will call a *counterexample algorithm*, because it is designed not for its intrinsic interest or practical value, but rather to serve as a counterexample to an impossibility conjecture. There are many other such examples in the impossibility result literature (some of which get picked on unfairly for not being practical).

As for Cremers and Hibbard, a lot of our work was devoted to formulating the model and correctness conditions. Their definitions were not sufficiently clean for us to be able to use them easily in our proofs. Our proofs involved constructing complicated bad executions; the properties comprising the problems statement are invoked repeatedly to justify the existence and properties of these executions. In order to use the properties in this way, we needed clean problem statements, so we had to simplify, generalize and polish the model. The details of the model description added a lot of overhead to the paper - so much overhead that it might serve as a significant obstacle for a reader.

The modeling considerations that arose in this work led directly to my own interest in formal models of concurrency, and especially in models that are suitable for use in impossibility proofs. In fact, the second piece of work I did in this area was the design (with Mike Fischer) of a general formal model for asynchronous shared-memory systems [81].

Soon thereafter, we obtained another collection of impossibility results [57, 53], this time for the *k-exclusion problem*, a generalization of mutual exclusion to some number $k > 1$ of resources. We showed that a strong simulation of a shared queue requires $\Omega(n^2)$ values of shared memory. We also obtained lower bounds for fault-tolerant versions of the problem, where the kinds of faults considered were just stopping faults. The techniques we used were similar to those in [26].

In [27], we considered what happens if memory is accessed via read and write operations rather than test-and sets. In this case, it turns out that mutual exclusion cannot be done at all using a single shared variable! It does not matter how many values the variable can take on. Moreover, this impossibility does not depend on fairness assumptions, but just on the two properties of mutual exclusion and continued system progress. More generally, n processes cannot achieve mutual exclusion with progress, with fewer than n separate shared variables). The proof again involves constructing incompatible admissible executions that look the same to some of the processes, so they behave incorrectly in some cases. This time, the key ideas are that (1) a process must write something in order to move to its critical region (to inform others), and (2) a writing process obliterates any information previously in the variable.

Using similar techniques, Rabin [92] proved a lower bound of $\Omega(n^{1/3})$ on the size of the range of test-and-set shared variables in any asynchronous shared-memory algorithm that solves the *choice coordination* problem. In this problem, processes share a common set of variables but do not have a common scheme for naming the variables; it is required that a marker be placed in exactly one of the variables.

2.2 Distributed Consensus

Around 1980, Leslie Lamport visited Georgia Tech, bringing along a copy of his new manuscript on the *Albanian Generals Problem*. Although superficially quite different from the resource allocation problems we had been working on, this problem had a similar "feel". As before, independent processes with separate inputs were required to accomplish some kind of coordinated action, in the presence of uncertainty about the rest of the system. In the case of distributed consensus problems, the uncertainty arises primarily because of the possibility of faults, rather than because of asynchrony. Local knowledge is again limited, this time not by bounds on the size of shared memory, but by the fact that all information must be conveyed via point-to-point message channels.

From the beginning, the area of distributed consensus has been a fruitful source of impossibility results. Some reasons for this are that the basic problem has a clean statement, and that there are many interesting variations of the problem, based on different assumptions about faults, timing, and kinds of agreement. The impossibility results in this area are based on just a few ideas, though. In what follows, I will group together results with related statements and techniques.

2.2.1 Number of Processes

The first group of results show how many processes are required to reach various kinds of consensus.

The first impossibility result in this area, the impossibility of reaching agreement among $3t$ processes in the presence of t Byzantine faults, appeared in the original papers [89] [73] on Byzantine agreement. The idea is based on processes "fooling" other processes, making them "believe" they are in different systems. The most pleasing proof I know for this result is not the original, but the *scenario* proof I did with Mike Fischer and Mike Merritt [54].

The following argument is for the case of $t = 1$, i.e., 3 processes and 1 fault. Suppose that p, q, and r comprise a 3-process solution that can tolerate 1 fault. Consider a system composed of two copies each of p, q and r joined into a ring, in order p_0, q_0, r_0, p_1, q_1, r_1. Let α be an execution of this system (a "scenario") in which each process with subscript 0 is started with initial value 0 and each with subscript 1 is started with initial value 1. Although the problem statement does not directly impose any conditions on scenario α, such conditions can be deduced.

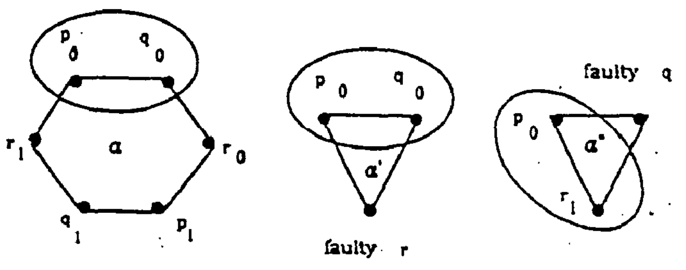

Figure 1: Scenarios for impossibility of consensus for 3 processes and 1 fault

Consider another scenario, α', consisting of one copy each of p, q and r, where both p and q are started with initial value 0. Process r is faulty in α', and sends to p exactly what r_1 sends to p_0 in α and to q exactly what r_0 sends to q_0 in α. In α', p and q behave exactly like p_0 and q_0 do in α, and receive exactly the same messages on their incoming channels. In α', p and q are required by the problem statement to eventually output 0; therefore, p_0 and q_0 will do the same in α. By similar reasoning, q_1 and r_1 eventually output 1 in α. However, it looks to p_0 and r_1 as if they are in another three-process scenario α'' in which q is faulty; the problem statement requires them to eventually output the same value in α'', and so they will also do so in α. This is a contradiction.

The idea of the proof in [89] is basically the same as in this example, except that instead of describing the scenario as the execution that is generated by a certain system started with certain initial values, Lamport et al construct the scenario explicitly. It seems to me that the higher level of abstraction of the [54] proof makes much clearer what is really going on. Perhaps there are other impossibility proofs containing explicit constructions of bad executions that could be made more understandable by describing the bad executions implicitly, by a simple way of generating them.

A related impossibility result for low connectivity networks appears in [39]; it says that at least $2t+1$ network connectivity is required to tolerate t faults. The proof is essentially another scenario argument similar to the one above (using a different scenario α).

Lamport also proved another impossibility result for 3 processes and 1 fault, this time for a weaker version of Byzantine agreement where the decision is only predetermined for executions in which no faults occur [72]. The proof in that paper is quite complex, but it is again essentially another scenario argument.

In his invited address at the 1983 PODC symposium [75], Lamport posed a problem about synchronizing clocks in a fault-prone distributed system. The processes are assumed to have separate physical clocks that can proceed at different rates; the object is for processes to compute adjustments to their physical clocks so that the nonfaulty processes' adjusted clocks remain close to each other (e.g., within a constant), and also so that they remain (approximately) within the range of the physical clocks. Dolev, Halpern, and Strong proved the impossibility of solving this problem with 3 processes and 1 fault [44]. I found this to be an immensely interesting result, but unfortunately I couldn't understand the proof; the main problem I had with it was that it was not based on a rigorous formal model. To help me explain the proof to my distributed algorithms class, I redid the proof using a scenario argument. (It was the need to redo this proof that led to the work in [54].)

The following is a very sketchy outline of the impossibility proof for synchronizing 3 clocks in the presence of 1 possible fault. Suppose p, q and r are processes that solve the problem. Consider scenario α composed of a large number of processes p_1, q_1, r_1, p_2, q_2, r_2, $\cdots$, arranged in a ring. The processes are supplied with physical clocks that run at constant rates, but the rates are different for different processes. The processes at one portion of the ring (say the top) have clocks that run slowly, while the processes at the greatest distance from the slow processes have clocks that run fast; in between, there are only tiny differences in rate between neighbors (but of course eventually the physical clocks of any two neighbors diverge).

Each pair of consecutive neighbors thinks it is in a 3-process scenario, so must synchronize clocks appropriately. Each neighboring pair ends up with adjusted clocks that are close in value. This requires either some slow processes to set their clocks far ahead or some fast ones to set them far back. Assume the former, without loss of generality. Then there are two slow neighbors that will set their adjusted clocks to be far ahead, which will take them out of the range of their physical clocks. But a comparison of α with a 3-process scenario that looks the same to these two neighbors shows that they must keep their adjusted clocks within the range of their physical clocks in α, a contradiction.

The paper [54] presents a collection of results about the number of processes and connectivity required for various consensus problems; these include the results just described. This was the first paper to organize the proofs using explicit and rigorous scenario arguments (although the same approach was implicit in the other papers I mentioned). As I said earlier, this approach is nice because it provides a high-level way of looking at the constructions, and because it unifies a lot of different-looking previous work. The paper does not contain one general theorem that implies all the results (which would be still better) but rather a general technique.

Some interesting modeling issues arose here. I usually like to present impossibility proofs using an explicit operational model, describing processes and the message system as some kind of state machines. Doing that for the ordinary or weak Byzantine agreement setting seems straightforward. But it is not clear what kind of model is appropriate for processes with physical clocks that move at different rates. It seemed at the time that if we gave all the details of such a model, it would be so complicated, and add so much overhead to the paper that no one would ever read it.

Our solution here was to give an *axiomatic* model (without saying what kind of mathematical object is supposed to serve as a model for the axioms). This approach tends to impose the fewest possible constaints on the system, making the result potentially applicable to more systems. On the other hand, such an approach is also potentially applicable to *no* systems - when proving impossibility results with an axiomatically-described model, one should be sure to check that some interesting models satisfy the axioms!

Another result that can be proved using the same techniques is the impossibility result proved by Karlin and Yao [68] for probabilistic Byzantine agreement using randomized algorithms. Knowing that n processes can't reach agreement with t faults when $n < 3t$, they asked with what probability such a small number of processes are able to agree. Their result shows that probability 2/3 is the best that can be achieved. Again, they used direct constructions of bad executions, but the proof can be done more simply using a scenario argument similar to the first one above.

It is an interesting open question whether this bound is tight (for symmetric Byzantine agreement algorithms, in which each process starts with an initial value), and how it extends to arbitrary values of n and t. Even though an answer to this open question may not have much direct practical significance, an answer to this question may give important insight into the power of randomized algorithms. (So far, there are very few results in the literature giving impossibility results for randomized algorithms.) Impossiblity results for some additional special cases of this problem are proved in the new paper [60] the proofs appear to be based on detailed analysis of the properties of randomized algorithms. The paper [40] extends the Karlin-Yao bound to hold even under certain restrictions on the power of the "adversary".

The paper [46] contains some lower bounds on the number of processes required to reach consensus in various fault and timing models. Proof techniques are based on scenarios.

The paper [31] contains lower bounds for the number of processes required to solve the Byzantine firing squad problem, using various fault and timing models. A nice touch here is that one of the results is proved by reducing weak Byzantine agreement to it rather than by a direct proof. For the other results, scenario arguments are used, this time based on a sequence of scenarios, $\alpha_1, \alpha_2, \cdots$; each successive pair of scenarios looks the same to some process, which therefore behaves in the same way in both cases. This leads to a contradiction when the constraints imposed by the problem statement are applied to some of the scenarios.

Thus, we have a collection of impossibility results for the number of processes and connectivity for consensus problems, all proved using scenario arguments. Several different kinds of models are used in this work. For the results about synchronous systems, the early work such as that in [89] used specially tailored formal models. The later work used more general and familiar-looking state machine models. These models are a lot simpler than those used for asynchronous systems, because the notions of timing and admissibility are much simpler. For the results about partially synchronous systems (e.g., the results on clock synchronization), it is not so clear what the proper model should be. Some of the proofs for partially synchronous systems are done informally and ambiguously. Some have very detailed and complicated special models, and some are done axiomatically.

2.2.2 Number of Rounds

My first reaction to Leslie's paper on Albanian agreement was that the clever algorithms in the paper ran too long! Surely, I thought, there must be a way to reach consensus in fewer than the $t+1$ rounds their algorithm required. (From my experience, this is most often the way impossibility proofs originate - one often doesn't start out thinking that the impossible task is impossible.)

Mike Fischer and I soon were able to prove a $t+1$ lower bound on number of rounds required for Byzantine agreement [56]. Our work on this result was another good example of the game of working on conflicting positive and negative results at the same time. We went back-and-forth, working alternately on algorithms and impossibility proofs, for several days. A counterexample arose for each algorithm we thought of, until finally one counterexample was extended to an impossibility proof.

The basic idea of the proof is pretty simple. Consider the case of two faults, i.e., where $t = 2$; we must show that two rounds can't suffice to reach agreement. We can assume without loss of generality that the algorithm consists of every process broadcasting its value, then repeatedly receiving messages from everyone and relaying everything that it received. So after two rounds, each process can record the information it has received as a matrix of values.

If a process sees a matrix of all 0's, it must decide 0, and similarly for 1. Also, it is possible to construct a *chain* of matrices, $M_1, M_2, \cdots, M_k$, starting with a matrix of all 0's and ending with a matrix of all 1's, where for each i, there is some execution with at most 2 faulty processes, in which some nonfaulty process sees M_i and some nonfaulty process sees M_{i+1} (so the decisions would have to be the same). This is a contradiction. The successive matrices in the chain can be constructed by converting one 0 entry to a 1 at each step, moving down the columns; at each step, two faults are necessary to produce an execution in which the two views can be presented to two processes.

This construction was done using an explicit construction of the executions; I don't know whether an implicit construction via a simple generator might be possible, as it was for the scenario work.

This lower bound was extended to the case where the processes participating in the algorithm are permitted to authenticate messages, in [43] and [37]. The proofs in those papers are also chain constructions; however, these constructions are much more complicated than the one in [56]. There is also some difficulty in defining what it means for a system to permit authentication of messages.

The lower bound was further extended to the case where the only kind of fault permitted was simply a stopping fault. Versions of this result appeared in several unpublished notes (by Hadzilacos, by Fischer and Lamport and by Merritt), so that it became part of the folklore, before it was finally written up by Dwork and Moses. They incorporated this work into their work on knowledge [47], believing that using explicit formal definitions of the "knowledge" that a process has during an execution would provide a helpful way of looking at constructions such as these chain arguments. (For example, if a process can see a certain matrix in either of two executions constructed for the chain in [56], we can say that the process does not "know" which of the two executions it's in.) It's still not clear to me whether or not the formal knowledge definitions help in explaining the combinatorial construction for the stopping fault lower bound; however, Dwork and Moses were able to generalize this lower bound to yield results for other problems of reaching "common knowledge" in synchronous systems. (In fact, they were able to do more, in particular, to analyze exactly which patterns of failures required the protocol to run for $t+1$ rounds.)

Moses and Tuttle extended the work in [47] to other fault models [86]. They obtained algorithms that terminate as quickly as possible in all executions; in fact, they were led to these algorithms by considering the impossibility results. (Along the way, they produced a simpler version of the $t+1$ round lower bound for stopping faults.)

Coan proved a $t+1$ round *worst-case* lower bound for consensus for randomized algorithms, assuming that no erroneous answers are allowed [34]. In this case, the result for deterministic algorithms carried over fairly easily. (For comparison, note that Feldman and Micali [52] have a new constant *expected time* randomized Byzantine agreement algorithm for the case where a small probability of error is allowed.)

Babaoglu, Stephenson and Drummond [17] showed similar lower bounds for models in which broadcast communication, rather than point-to-point communication, is used. Their bounds depend on the "broadcast degree".

The paper [36] contains a lower bound for the number of rounds required for distributed processes to reach approximate agreement on a real number (rather than exact agreement on a value). A chain argument is used to show that no approximate agreement algorithm can converge too fast, in the case of Byzantine faults: for any k-round approximate agreement algorithm, there must be some executions such that the ratio of the range of output values to the range of initial values is at least $(t/nk)^k$.

The simplest style of approximate agreement algorithm, one that repeats a simple 1-round averaging algorithm k independent times, does not meet this bound, but rather achieves a ratio of around $(t/n)^k$. (It converges more slowly than the lower bound indicates). Another lower bound in [36] shows that this is the best that can be achieved by an algorithm with such a round-by-round structure. The argument is another chain argument.

These impossibility results left open the question of whether a better algorithm might be possible if it were not required to be round-by-round. Fekete answered this question positively [50], giving a clever counterexample algorithm that uses information from prior rounds: some fault detection is carried out and then the results of processes known to be faulty are ignored. This is one of the first examples where detection of Byzantine faults was shown to lead to improved results; it came about because of an impossibility conjecture.

Fekete's work in [50] and [51] contains lower bounds on the rate of convergence for crash and omission fault models, analogous to those for Byzantine faults. Again, chain arguments are used.

Thus, there are many lower bound results for the number of rounds required to solve consensus problems, all based on chain arguments. The kinds of models used here are primarily fairly straightforward synchronous state machine models, augmented in some cases with knowledge definitions.

2.2.3 Number of Messages

Dolev and Reischuk [42] proved lower bounds on the number of messages and number of signatures required for Byzantine agreement algorithms that use authentication, using scenario-style arguments.

2.2.4 Asynchronous Impossibility Results

So far, the bounds I've described for consensus protocols have been mainly for synchronous algorithms, and they have all been quantitative (lower bound) results. There has also been a lot of work on absolute impossibility results for purely asynchronous algorithms.

The "Two Generals" result in [61] should probably be classified as the first impossibility result for consensus in an asynchronous distributed system, although it isn't so much the asynchrony that is important here, but rather the uncertainty of message delivery. This result says that it is impossible for two distributed processes communicating via an unreliable message system to reach consensus.

The proof presented in [61] is pretty informal; when I worked it out formally it looked like a chain argument, but of a slightly different sort from the chains constructed for the round bounds.

Starting from an execution in which both processes decide, say on value v, a chain of executions is constructed by successively removing the last message receipt event. Each pair of consecutive executions looks the same to one of the processes, and the two processes must decide on the same value in each execution; it follows that a decision of v is reached by both processes, in all the executions in the chain. Among the executions in the chain is a "null" execution in which no messages are ever received; starting from this null execution, the chain can be further extended to produce another null execution in which neither process starts with initial value v, and yet a decision of v is reached by both processes. Under some reasonable assumptions about initial values and their relationship to the final decisions, it can be shown that such an execution should not result in a decision of v.

A similar argument is used by Koo and Toueg [69] to show the impossibility of achieving any knowledge gain in an asynchronous network, in the presence of even transient communication failures.

Halpern and Moses [64] have used formal notions of knowledge to describe the result of [61]. They also show that, in a precise sense, common knowledge cannot be gained in an asynchronous system. The techniques are basically similar to Gray's. Chandy and Misra [29] also show a similar result.

The next impossibility result I know about for asynchronous consensus is my result with Mike Fischer and Mike Paterson in [55]. This result shows the impossibility of reaching consensus in asynchronous systems, even when the message system is reliable, and even if the processes communicate via broadcast primitives, if there is the possibility of even a single stopping fault.

Just as for the $t + 1$ lower bound on rounds, we began our work on this problem by guessing that a solution was possible (for t faults, if n was sufficiently large relative to t). We had already had experience extending some synchronous agreement algorithms to the asynchronous setting; in the asynchronous setting, processes can wait to hear from all but t processes, so adding some extra processes sometimes permits an algorithm to compensate for the uncertainty of the missing messages. Again, we worked on both directions alternately, until the final result arose from a counterexample.

The version of our proof that I like best was developed by Bridgeland and Watro; similar ideas appear in recent work of Taubenfeld, Katz and Moran [98].

For simplicity, we restrict attention here to Boolean values only.

If v is a Boolean value, we say that a configuration C is "v-valent" if v is the only possible decision value reachable from C; we say that C is "bivalent" if both values are reachable. First, it is shown that any asynchronous consensus protocol that is resilient to a single fault has a bivalent initial configuration. Next, it is shown that any asynchronous consensus protocol that has an initial bivalent configuration and that works correctly when there are no faults must have a reachable configuration C in which there is a *decider* process p. This means that from C, it is possible for p to take some finite sequence of steps leading to a "0-valent" configuration, and also some other finite sequence of steps leading to a "1-valent" configuration; that is, p can make the decision on its own.

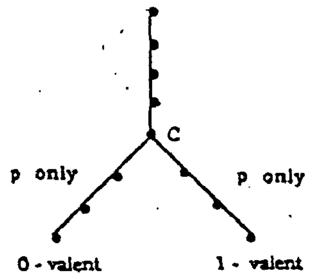

Figure 2: A decider process

The reason this is true is roughly as follows. The problem statement implies that we can't have an infinite execution consisting of bivalent configurations in which all processes continue taking steps and all messages eventually get delivered. Therefore, there's a reachable bivalent configuration C and a particular message m in the message system such that any configuration resulting from delivering m is univalent. Then there are two "neighboring" configurations D and E (that is, one a child of the other) such that delivering m from one leads to 0-valence and the other leads to 1-valence. This can only happen if the "neighbor edge" corresponds to a step of the same process p that is the recipient of m. But this means that p is a decider.

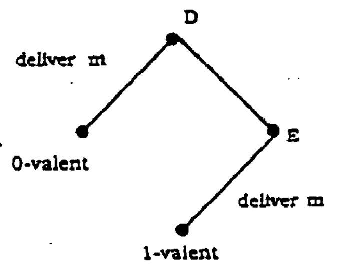

Figure 3: Neighboring configurations D and E leading to configurations of opposite valence

But such an algorithm with a decider, say p starting from configuration C, cannot be resilient to a single fault. This is because the rest of the system, operating on its own starting from C, is required to decide either 0 or 1, but it can't tell whether p has already decided differently.

Here again, as for the shared memory work and all other work on asynchronous algorithms, it is important to be careful about stating and using admissibility assumptions (the liveness assumptions about how the system runs). Here, the admissibility assumptions are that the non-failed processes continue to take steps (as long as there are steps to be performed), and that all messages eventually get delivered. It is possible to get much easier proofs, for example, if messages are not required to be delivered; one such proof is given in [28].

Our original proof was similar to this one, but it turned the ideas around; we assumed the existence of a resilient algorithm and arrived at a contradiction. As usual, the contradiction involved constructing a bad admissible execution. The new proof organization is better because it is not just a proof by contradiction, but also gives some positive information about (non-fault-tolerant) asynchronous consensus protocols.

The general technique used here is to analyze the ways in which the system configuration can move from being bivalent to being univalent, showing that none of them can work properly in all cases. Or, turning the proof around, starting with a bivalent configuration, construct an admissible execution in which the configuration stays bivalent (by analyzing the ways in which decisions are made).

The modeling issues were interesting here. As in the earlier papers on shared memory, this paper contains a carefully-developed formal model for asynchronous computation, but this time specially tailored to message communication. The model isn't very complicated, but it is a little annoying that the modeling work starts from scratch, borrowing nothing from previous work in model development for asynchronous shared memory systems. Since both kinds of systems deal with ideas such as admissibility and control of actions, it seems that a common foundation could have been used. It would be very nice if there were some body of common definitions that people could use for asynchronous computing impossibility results, that would remove some of the overhead of the model section of each paper.

Another problem with the model in this paper is that some of the particular aspects of the model, such as the particular protocol used by the nodes in interacting with the message system, seem very special. Perhaps a more general model could have been useful here, in trying to apply this result to slightly different settings. I have since redone this proof for my class using the general "I/O automaton" model for asynchronous concurrent computation, (defined in [79, 80]), but I can't yet tell how much of an improvement this is.

This paper, unlike most impossibility results, has had lots of attention, even from practitioners. For example, one system designer's reaction was to say "of course" - this theorem formalized an intuition that she had already had about distributed systems. This doesn't mean that she "knew" the result in the sense that theoreticians mean (that they can prove it), but rather that her experience with things like commit protocols had led her to believe that this kind of thing could not be done. She probably would not have known *exactly what* couldn't be done - formulating the precise assumptions on which such results depend is the sort of delicate analytical task that will probably always be left to the theoreticians.

Distributed system designers do seem now to be generally aware of the limitation expressed by this result. I sometimes hear people describing their system designs by saying that the system cannot achieve a certain behavior because of this known limitation; they then go on to describe the weaker guarantees that their systems do make. In one case, system designers were surprised that what they believed their system was guaranteeing was in fact impossible; they did not give up on their project, though - rather, they used the new knowledge to help them clarify their claims about what their system did. So it seems that this impossibility result has had the beneficial effect of helping (or forcing) system designers to clarify their claims about their system.

This result has seen some follow-on technical work. There have been positive results, including counterexample algorithms for variations of the problem and some algorithms that may be interesting in themselves. There have also been related impossibility results.

For example, Ben-Or [19] and later Rabin [91] devised interesting randomized algorithms that circumvent the impossibility result; these algorithm eventually decide with probability one, and never violate safety properties. Also, Dwork, Lynch and Stockmeyer [46] devised consensus algorithms for the case where the problem definition is weakened to allow nontermination if certain nice timing conditions (i.e., upper bounds on message delivery time) fail. (An interesting technical open question remains about the time requirements for consensus in the model of [46].)

Dolev, Dwork and Stockmeyer [41] noticed that there were several different kinds of asynchrony in the execution model of [55], e.g., asynchrony of messages and of processes. They classified systems, based on the various combinations of these factors, and obtained impossibility results for many of these cases (and algorithms for some). The impossibility results were proved using bivalence arguments similar to that in [55].

Their proofs proceed by contradiction, constructing bad executions; these executions had to be carefully designed not only to satisfy admissibility assumptions like the ones for asynchronous systems, but also to satisfy additional requirements of partial synchrony as required by the various cases. They analyzed the ways in which decisions must be made, e.g., locally to a single process, and showed that none of them can work correctly in all cases, e.g., a resilient protocol must be able to proceed without the deciding process. In some cases, they obtained impossibility of 2-resilient consensus, rather than 1-resilient consensus (because analysis of the decision point showed that the rest of the system might have to proceed without 2 processes, in order to produce a contradiction).

The models used in [41] are quite detailed and specialized, for example, in their assumption that time has a minimum granularity; it makes me think that a more abstract or general model could have been used here. For example, some recent work on impossibility results for atomic register problems seems quite similar to some of the work in [41]; if the earlier work were stated in a more general way, perhaps it would imply some of the new register results.

Attiya, Dolev and Gil [9] extended some of the work in [41] to consider Byzantine faults, not just stopping faults. They considered asynchronous processes with time-bounded communication. They gave an impossibility result that shows that a process cannot be guaranteed to decide in a bounded number of its own steps. The argument is a simple bivalence argument, similar to arguments in [41]. They also proved a $3t$ vs. t process impossibility result, using a scenario argument.

Welch [100] presented a nice reducibility argument that yields one of the main impossibility results of [41] directly from the result of [55]. The reducibility uses a fault-tolerant version of Lamport's distributed clock idea [74].

Moran and Wolfstahl [85] gave two generalizations of the result of [55], one using a similar proof and one using a reducibility from the result of [55]. They defined two graphs to represent the problem being solved: an *input graph* for the possible input vectors, and an *output graph* for the allowable decision vectors. (In each graph, an edge joins two vectors if they differ in exactly one component.) Their results show the impossibility, in the presence of one faulty process, of performing any task that has a connected input graph and a disconnected decision graph.

Bridgeland and Watro [24] presented more impossibility results generalizing [55], using similar proofs. Their work also originated the notion of a "decider", described above.

Attiya, Bar-Noy, Dolev, Koller, Peleg and Reischuk [10] presented three impossibility results for the "process renaming problem", wherein anonymous processes that start with distinct names from a large ID space are supposed to decide on distinct names from a fixed (smaller) ID space. They showed that renaming with n names is impossible; the argument is again very similar to that of [55], using generalizations of the sort obtained in [85]. (Some special argument is needed to show the bivalence of an initial configuration.) There is a very interesting open question remaining here, about whether the lower bound for t faults can be extended from $n+1$ to $n+t$ names. An algorithm in their paper shows that $n+t$ names suffice.

They also showed that the problem cannot be solved with $2t$ processes, using a simple scenario argument. Finally, they considered an order-preserving version of the problem, which they showed to have an interesting and large lower bound (for which they have a matching upper bound). The reason so many names are required is that processes sometimes have to decide on names for themselves while there are still

t silent processes; in this case, they need to reserve enough space for all possible relative orderings among those processes and between those processes and the others.

Biran, Moran and Zaks [20] extended the [85] work to a characterization of what can be done with one faulty process. Their characterization has a pleasing graph-theoretic flavor, in terms of input graphs and decision graphs. More specifically, they proved a variant of the [85] impossibility results, plus a new result giving a second graph-theoretic condition implying impossibility. On the other hand, they were able to obtain a protocol for the case where both of these conditions fail. They also utilized the graph characterization to get a lower bound on the number of messages require. Taubenfeld, Katz and Moran [98] have made preliminary attempts to extend the work in [85] to characterize what can be done in the presence of t faulty processes.

Loui and Abu-Amara [76] proved results about the impossibility of resilient consensus in shared-memory rather than distributed models, in case the allowable operations on shared memory are reads and writes, and also in case they are test-and-sets. The ideas used here are very similar to those used for the related distributed results. The similarity between the ideas used in these two settings reinforces my intuition that there is an awful lot that is fundamentally the same in the two environments.

They proved the impossibility of 1-resilient consensus for read-write shared memory. The construction is very similar to that in [55] and [41]: a bivalence argument with a simple case-analysis about the decision point. They also proved impossibility for 2-resilient consensus for test-and-set shared memory, in the special case of binary values. This is another bivalence argument, but, as in [41], this time analysis of the decision point shows that all except 2 processes might have to proceed in order to produce a contradiction. Both of these results extend immediately to fully resilient algorithms - algorithms that tolerate arbitrary numbers of faults - a fact that was useful in the work on atomic registers that I will describe later. Chor, Israeli and Li [30] also proved the first of the two impossibility results in [76].

In the asynchronous consensus work based on shared memory, admissibility considerations are simpler than they are in the distributed work. Here it is only necessary to ensure that (non-failed) processes continue to take steps, it is not necessary to worry about ensuring message delivery.

Thus, there are many closely related results that describe what cannot be done in fault-tolerant asynchronous systems. Nearly all of these results are proved using similar bivalence arguments.

2.2.5 Commit

The commit problem is a particular kind of binary-valued consensus problem, where the two values are known as "commit" and "abort". This problem requires agreement and termination; in addition, a "commit rule" should be satisfied, e.g., saying that if any initial values are "abort" the decision must be "abort", while if all initial values are "commit" and there are no failures, then the final result is "commit". The impossibility result of [55] implies that the commit problem cannot be solved in an asynchronous setting, so it is usually considered in synchronous and partially synchronous models.

Dwork and Skeen [48] considered the commit problem in a synchronous complete network model. They proved a lower bound of $2n - 2$ messages for every failure-free execution that results in a commit decision. This proof is based on a simple argument that there must be a path of messages from every process to every other (or a wrong decision could result). This proof was redone using formal notions of knowledge by Hadzilacos [62]. Dwork and Skeen also proved lower bounds on the number of rounds required in failure-free executions, based on the same fact (the existence of paths of messages between pairs of processes) and an assumption about bounded bandwidth. Segall and Wolfson [96] generalized the Dwork-Skeen message bound result to give a lower bound on the number of message hops needed for solving the commit problem in incomplete networks.

Coan and Welch [33] considered the commit problem in a partially synchronous model, for randomized algorithms for which eventual termination is required with probability 1. Also, a commit decision is only required in case all processes have initial value "abort", the execution is failure-free and all messages are delivered within a fixed bound time b that is known to the processes. The main point of their paper is actually an upper bound result: a fast randomized commit protocol for $n > 2t$; in order to argue that this protocol is close to optimal, they showed two limitations. First, they showed that no solution is possible if $n \leq 2t$. This proof does not seem to be a scenario argument like the other proofs of lower bounds on the number of processes (at least not obviously); it's a complex explicit construction of bad admissible executions. (Perhaps a higher-level argument might be possible.)

The second impossibility result in [33] says that it is not possible for each process to make a decision within a bounded expected number of its own steps. (The time bound for the protocol in the paper is measured in terms of a non-local asynchronous round measure.) The proof of this result is a bivalence argument that allows construction of not just a single non-deciding execution, but of lots of long nondeciding executions. (This is in order to obtain a bound for the average.) This is an interesting extension of the bivalence technique. Note that the two impossibility results of [33] are among the very few impossibility results that make interesting claims about randomized protocols.

2.2.6 Synchronization

I am here grouping certain synchronization problems together with the consensus problems, since they involve processes agreeing on when to perform actions.

Arjomandi, Fischer and Lynch [8] proved a lower bound on the time for an asynchronous, reliable network to carry out a simple synchronization task - to perform s "sessions", in each of which all the processes in the network must perform at least one output event. The result is a lower bound of approximately sd on the time for performing s sessions, where d is the diameter of the network. Since a synchronous system would only require time s, this amounts to a provable difference in the time complexity of synchronous and asynchronous systems.

The proof idea is simple. First, note that an execution can be represented by a diagram with time lines for processes and connecting edges for messages. These time lines and connecting edges represent dependencies among events. Such a diagram can be "stretched" without violating the dependencies, and processes will not be able to tell the difference. Now, if an execution takes too little time, it can be partitioned into $r-1$ short intervals, $int_i, 1 \leq i \leq r-1$, in each of which there is insufficient time for a message to propagate between a certain pair of processes, p_i and q_i. Then it is possible to modify the execution by stretching its diagram so that all steps of p_i follow all steps of q_i in interval int_i; if the p_i and q_i are chosen appropriately, the modified execution will not contain r sessions.

This result demonstrates that lower bounds can be proved on time, even in asynchronous networks. This is not usually done, but I see no good reason why not. Appropriate ways of measuring time are available for asynchronous systems, such as those defined in [90],[81],[79]. and [83]. Proving such lower bounds is a good area for future research.

Awerbuch [16] proved a time/communication tradeoff lower bound for any *network synchronizer*, i.e., a program designed to adapt synchronous algorithms for use in (reliable) asynchronous networks. The techniques are generally similar to those used in [8], and are based on the necessity of communication between various pairs of nodes between pulses of the synchronous algorithm being simulated. This proof, however, uses some fancier graph theory.

Lundelius and Lynch [77] proved a lower bound on how closely software clocks of (nonfaulty) distributed processes can be synchronized, in terms of the uncertainty in the message delivery time between pairs of processes. In particular, we obtained an interesting tight bound of $2\epsilon(1-1/n)$ for complete graphs. The idea is to represent an execution by a diagram as in [8], but with message edges tagged with message delivery times. This diagram can be "stretched" as before, but this time keeping the new message delivery times within the allowable bounds, and everything will still look the same to all the processes. Applying inequalities representing the constraints of the problem to the various stretched diagrams gives a contradiction.

Dolev, Halpern and Strong [44] gave a lower bound similar to that in [77], but characterizing the closeness of synchronization obtainable along the0 real time axis. That is, they proved a lower bound on how close the real times can be when two processes' adjusted clocks have the same value, whereas our result is a lower bound on how close the adjusted clock values can be at the same real time.

Halpern, Megiddo and Munshi [63] extended the results of [77] to other kinds of graphs besides just complete graphs, using the same basic kind of stretching arguments. (The characterization for general graphs is not as nice as for complete graphs, however.)

2.3 Shared Registers

Now I reconsider shared memory asynchronous algorithms, in a setting similar to the one I started this talk with. In the past couple of years, there has been a lot of interest in problems about implementing different kinds of shared registers in terms of other kinds of shared registers, generally in a "wait-free" manner.

Lamport's paper [71] includes one impossibility result - a result that says that atomic registers cannot be implemented in terms of regular registers unless the readers write. The proof is based on a new axiomatic partial ordering model introduced in [71]. The proof is only sketched, and involves an explicit construction of bad executions. Although the result is probably correct, I do not believe that it actually follows as claimed from the axiomatic model given in that paper.

Herlihy's interesting paper [65] contains impossibility results and also universality results. He noted a connection between (fully-resilient) consensus results and the (wait-free) implementation of atomic registers. Namely, if one type of data object can implement fully resilient consensus and another cannot, then the first data object cannot be implemented in terms of the second, in a wait-free manner. (There is a close connection between the full resiliency property and the wait-free property, although they originated in different contexts.) In particular, the objects I described above in connection with [76], (read-write objects and binary test-and-set objects) plus others described by Herlihy, cannot provide wait-free implementations of objects with more powerful operations such as general test-and-set.

The proofs are bivalence arguments, but they are actually somewhat simpler than the proofs in [76], because the notion of admissibility used here is less restrictive than that used in the results on 1-resilient and 2-resilient consensus. The full resiliency assumed here means that the only liveness condition needed for admissibility is that some process continue taking steps, i.e., that the execution be infinite. It is somewhat easier to construct infinite non-deciding executions than non-deciding executions satisfying some extra admissibility conditions.

Again turning the proof around in the style of [24], implicit in this work is a lemma that says that fully resilient consensus implies the reachability of another kind of "decider" configuration: one that is bivalent but for which any step of any process leads to a univalent state (in one step). This is a different notion of a decider from the one used by Bridgeland and Watro; theirs involves a *particular* process forcing either of two different decisions in some number of its own steps, whereas Herlihy's means that *any* process can force a decision in one step. This simplified notion of decider leads to simpler proofs here than in [24] and [55].

Thus, the bivalence technique is useful (indirectly) in getting more than just consensus impossibility results. Here, reducibilities show its utility in proving that some kinds of objects can't be implemented in terms of other kinds.

Some interesting modeling issues arise. For example, Lamport's impossibility proof sketch in [71] is based on his axiomatic partial ordering model. Herlihy's work, on the other hand, uses I/O automata. His method of using I/O automata to model registers differs from the way they are used to model registers in [21] and [95] Herlihy uses special "scheduler" machinery not used in the other work. Some work still seems needed to determine the best way to use this general model to describe registers. It is also not clear whether the axiomatic model or the I/O automaton model is better for describing these results, or whether the two should somehow be combined.

The I/O automaton model is not often used for reasoning about shared memory algorithms. This is because that model handles input and output events separately; for reasoning about shared memory algorithms, one would often like to avoid handling these two kinds of events separately, treating an invocation of an operation on a shared objects and a corresponding response as indivisible. (For example, the models in [81] and in [76] do this.) However, in the work on wait-free shared registers, It is appropriate to handle these two events separately, making I/O automata a reasonable model. The major point about atomic objects is that they make it appear "as if" accesses were performed indivisibly; this suggests that it might be useful to have two models (or two instances of one general model), one like [81] in which the accesses are indivisible and one like I/O automata in which they are not; connections between the two models should be proved.

It is still not clear to me what the proper formal definition of the "wait-free" property should be. Perhaps it should be defined (as in [65]) in terms of a bounded number of process steps, perhaps in terms of an asynchronous time measure, and perhaps in terms of failure resiliency. This needs more work.

2.4 Computing in Rings and Other Networks

Now I switch to another area in which the proofs are very different from the ones I have considered so far. This area contains many impossibility results, most involving the message cost of carrying out various computations in a network. The case most commonly studied is that of a ring network.

Some of these results are based on a *distance* argument: in a ring, it takes many messages to get information from one place to another. Another basic idea is *symmetry*. For instance, a ring containing indistinguishable processes is a very symmetric configuration; if it is to accomplish a task involving breaking symmetry, some process p must send a message; then because of symmetry, all processes indistinguishable from p will also send messages.

There are so many results in this area that I couldn't really classify them very well. Many of the results seem related; for instance, there are many results giving lower bounds of $\Omega(n \log n)$ on the number of messages required to solve certain problems in a ring. Some work still seems to be required in unifying these results.

2.4.1 Absolute Impossibility Results Based on Symmetry

The earliest paper giving impossibility results in this area seems to be the very interesting paper of Angluin [7], which proves the impossibility of electing a leader in various graphs. The processes in her model are indistinguishable, and they have no inputs, so all that can be used to distinguish them is their position in the network graph. But many graphs have symmetries that will prevent a guarantee of distinguishing any process - anything that one process can do, the others symmetric to it might do also. The paper identifies symmetry properties of graphs that lead to impossibility of leader election. This paper can be credited for the now well-known and simple folk theorem that says that it is impossible to elect a leader in a ring (with a non-randomized algorithm), if processes do not have unique ID's.

One unusual feature of this paper is that it uses a model based on Hoare's CSP. This is the only example I can think of, of CSP being used for an impossibility result. It has many features that seem to me to be too distracting for such proofs.

Johnson and Schneider [67] gave impossibility results related to Angluin's for several different problems using several different models; the models are based on CSP, read-write shared memory, and variables with locks. Other related results appear in [23].

2.4.2 Lower Bounds for Rings

Many lower bound results have been proved expressly for ring networks.

Burns [25] proved an $\Omega(n \log n)$ lower bound on the number of messages required to elect a leader in an asynchronous ring. The key idea is the limitation of local knowledge based on how far information can travel - it takes k messages to propagate information to a process distance k away. The proof does not require any special assumptions about process ID's: processes can have distinct ID's chosen from any ID space.

Roughly speaking, Burns' proof shows inductively on n that there are a large number of segments of length n each of which is capable of generating $\Omega(n \log n)$ messages on its own (without any communication arriving from the endpoints). For the inductive step, suppose there are many segments of size $n/2$ each of which can generate lots of messages, and try to get some of size n that also can generate many messages. Suppose they don't exist. Consider all possible ways of concatenating pairs of the segments of size $n/2$. If such a double segment is unable to generate lots of messages on its own, then consider an execution in which the two halves first quiesce, then some additional messages flow starting at the merge point. Because of the limitation on number of messages, information about the merge cannot propagate as far as the middle of either of the two halves before the double segment quiesces.

So this means that there must be a large set S of length $n/2$ segments such that any double segment composed of segments in S quiesces without information about the merge propagating as far as the midpoint of either half. Now consider what happens when any number of segments in S are formed into a ring. They have executions in which the length $n/2$ segments quiesce first, then the additional messages propagate from merge points (but not as far as the midpoints of the S segments), until quiescence occurs. This means that each individual process' decision can only depend on local information: information about its own S segment and about its nearest adjacent S segment. But then inconsistencies can arise based on different ring arrangements: the fact that some of these rings elect a leader implies that others can elect more than one leader.

Much attention is devoted in this paper to the design of an appropriate formal model for message-passing systems, as is suggested by the paper's title.

Pachl, Korach and Rotem [87] extended the $\Omega(n \log n)$ lower bound of [25] to the *average* case, for asynchronous deterministic leader-election algorithms. The techniques are similar. They also proved a lower bound for unidirectional rings in which processes are interrupt-driven, using a different style of argument based on the special structure of such algorithms. Such algorithms are essentially deterministic; what happens at each process can be viewed as a transformation from input strings to output strings. Pachl [88] extended the results of [87] to the case of randomized algorithms where a nonzero probability of erroneous outputs is permitted. Related results were proved by Duris and Galil [45] and Bodlaender [22].

Burns' proof depends heavily on the asynchrony; for instance, construction of bad executions involves forcing subsegments to quiesce separately, then to quiesce around the merge points. Frederickson and Lynch [58] considered the same problem for synchronous rings. In the synchronous case, the absence of a message might be regarded as a special "null message", and used to communicate something. We showed that this apparent extra capability doesn't help - an $\Omega(n \log n)$ lower bound still holds.

Now special restrictions are needed on the algorithm in order to obtain the lower bound. Namely, the algorithm is required either to be comparison-based, or to use a very large ID space relative to the running time. The result for the second assumption follows by a reducibility from the result for the first, by a Ramsey's Theorem argument; the argument says that with enough ID's, the algorithm must behave like a comparison algorithm on some subset of the ID's.

The idea of the first proof is that many messages are required to break symmetry. Consider for example the ring consisting of processes with ID's 0,4,2,6,1,5,3,7.

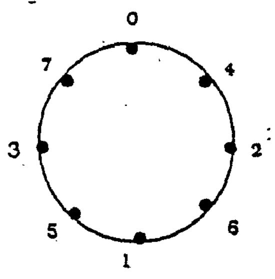

Figure 4: A symmetric ring of size 8

This ring is very symmetric, up to comparisons. In particular, adjacent segments of length 2^k are comparison-equivalent. So everything looks the same (up to comparisons) to processes 2^k apart until some chain of (real, not null) messages spans a distance of at least 2^k. Until then, if one process sends a message, so does every process a multiple of 2^k away.

It is easy to produce highly symmetric rings of size equal to any power of 2. Much of the effort in the paper is devoted to producing highly symmetric rings when n is not a power of 2.

This paper also contains a counterexample algorithm. This algorithm shows that you cannot remove all assumptions about ID's or running time. For otherwise there is a very time-consuming algorithm (its time complexity depending exponentially on the ID's actually in use) with only $O(n)$ messages. This algorithm does not seem to be very interesting in itself, but it is interesting because it demonstrates the need for the assumptions in the lower bound.

Attiya, Snir and Warmuth, in [14] used similar ideas to those in [58] but took them much further. They considered the case where there are no ID's built in, but (for certain problems) processes may start with input values. The object is for the processes to compute some function (invariant under circular shifts) of the input vector.

They considered both asynchronous and synchronous rings. For the asynchronous case, they obtained an $\Omega(n^2)$ message lower bound for many interesting computable functions, including AND and MAX. This bound contrasts with the $\Theta(n \log n)$ bounds that hold for the case where the processes have distinct ID's.

The proof involves constructing a "fooling pair" of rings, R_1 and R_2, where R_1 is very symmetric, but R_2 need not be, together with a process p which has a big neighborhood that is the same in both rings but which is required to decide differently in the two rings. Then messages must propagate to p from outside the common neighborhood (in both rings). However, when a process sends a message in R_1, many symmetric processes must also send messages.

The proof for the synchronous case uses a similar argument, but now it only yields a lower bound of $\Omega(n \log n)$, because of the possible utility of null messages. Now a stronger definition is needed for a fooling pair, in which both R_1 and R_2 must be very symmetric. Then it can be shown that the algorithm causes many messages to be sent in both R_1 and R_2.

The lemmas used in [14] are slightly different from those used in [58]; instead of analyzing chains of messages in detail, they are stated in terms of less detailed information about the number of rounds at which some message is sent. As in [58], much effort is devoted here to producing the strong symmetries needed for rings whose sizes are not powers of two.

Several other recent papers contain results related to those in [14]. Moran and Warmuth [84] proved a lower bound of $\Omega(n \log n)$ for the number of bits required to compute any "nontrivial" function on a deterministic ring with indistinguishable processes. Attiya and Mansour [12] gave a proof that any "non-quasi-permutation-free" regular language requires $\Omega(n \log n)$ messages, using the synchronous theorem from [14]. Attiya and Snir [13] considered the *average* case for deterministic algorithms, in the asynchronous setting. They showed a lower bound of $\Omega(n \log n)$ for the average number of messages required by any deterministic algorithm for computing an arbitrary "nonlocal" function. Roughly speaking, they showed that there are many sequences of processes of any given length k in which messages are sent by the center process in the sequence at round k (in a synchronous execution of the algorithm); this implies that many messages are generated in an "average ring". This lower bound extends easily to randomized algorithms that admit no probability of error, using a simple reduction. However, if nonzero error probability is allowed, then the lower bounds fail (and an $O(n)$ algorithm exists).

Abrahamson, Adler, Higham and Kirkpatrick proved a collection of amazing lower bounds for randomized algorithms for solving certain problems, e.g., "solitude detection", in a ring. They allowed a nonzero probability of error, and measure the communication bit complexity. The model is asynchronous, but it's unidirectional and interrupt-driven, so (as in [87]) its behavior is very constrained.

They studied many different cases, e.g., in which the ring size is either known or unknown, and in which decisions are revocable or irrevocable. The lower bounds are quite complicated-looking functions, but what's most amazing is that they are tight.

One key idea is the following. If the expected cost of computations in a particular ring is low, then for some fixed boundary in the ring, and for some fixed short sequence of messages, computations having that sequence at that boundary occur with reasonably high probability. Then it is possible to splice together multiple copies of that ring, by cutting and splicing at the designated boundary. Then with reasonably high probability, solitude will be verified erroneously in the spliced ring. This argument can be thought of as a sophisticated form of symmetry-breaking, incorporating ideas reminiscent of *crossing sequence* arguments in Turing machine theory. (Some of the techniques used in this work also extend to proving lower bounds on the *best case* bit complexity for a nondeterministic algorithm.)

The model definitions are an important part of this work, because the results are very sensitive to slight variations in assumptions. Unfortunately, these definitions do impose a lot of overhead on the reader. This work contains different sets of problem statements, strong ones for the algorithms and corresponding weak ones for the impossibility results, thus making each result as strong as possible.

Mansour and Zaks [82] considered the case where the ring starts with a leader, but where the ring size is unknown. Even with a leader, interesting lower bounds still hold for other reasons. They showed that recognition of any nonregular set requires $\Omega(n \log n)$ bits of communication.

Finally, Goldreich and Shrira [59] proved an $\Omega(n \log n)$ lower bound on the number of messages for function computation in an asynchronous ring in which one link might fail, even if the ring has a leader and the ring size is known. The basic idea is that the leader needs to hear from everyone; to ensure this, it must initiate messages in both directions, which need to propagate until they reach the broken link (if any), and then responses must come back. But a node doesn't know if it's adjacent to a broken link; to be safe, it might have to behave as if it were even if it is not, and send messages back toward the leader. This means that the leader might get messages reflected back from "fake extremities" and still not have heard from all processes. In that case, the leader needs to initiate messages again. The paper [59] also contains an $\Omega(n^2)$ lower bound for the case where the ring size is unknown.

Thus, I have described a collection of bounds for ring computations that depend mainly on symmetry, on the distance messages have to travel, and on crossing sequence arguments. It seems to me that there is some good work still to be done in coalescing, generalizing and simplifying this work.

2.4.3 Lower Bounds for Complete Graphs

Some lower bounds on the number of messages have also been proved for complete graphs. Korach, Moran and Zaks [70] proved tight lower and upper bounds for some distributed problems in a complete asynchronous network of processes. They obtain $\Omega(n \log n)$ lower bounds for leader election and spanning tree determination, and $\Omega(n^2)$ for certain matching problems. Afek and Gafni [3] proved similar bounds to those in [70], for leader election; theirs, however, extend to the synchronous case, and they also prove time bounds.

2.4.4 Lower Bounds for Meshes

Abu Amara [2] showed a lower bound of $(57/32)n$ on the number of messages required for comparison-based leader election in a synchronous mesh consisting of n nodes.

2.4.5 Lower Bounds for General Graphs

Other related bounds have been proved for general graphs. Santoro [94] proved a lower bound of $\Omega(n \log n + e)$ for leader election in general graphs. The $n \log n$ component results from the corresponding bound for rings. The e component is based on a "folk argument" that all edges need to be traversed, in order to ensure that no other nodes are hidden in the middle.

Awerbuch, Goldreich, Peleg and Vainish [15] proved a very nice lower bound that says that it's necessary to "involve" all the edges in a network in order to solve certain problems, such as broadcast communication, election, constructing a minimum spanning tree, or counting the number of nodes in the network. This implies that the number of fixed-length messages needed is at least e. The argument is for comparison-based algorithms, but can be extended to more general algorithms using Ramsey Theory techniques similar to those used in [58]. This result builds on an earlier weaker result by Reischuk and Koshors in [93].

The result follows as in [94] in case no one knows identity of neighbors, so this work supposes that each node knows the identity of its immediate neighbors. Then it cannot be proved, as in [94], that a message actually gets sent on each edge; however, bad executions based on duplicate graphs with pairs of crossover edges demonstrate that a node must somehow find out additional information about its neighbor; roughly speaking, this involves the node receiving a message containing the identity of the neighbor, to use for comparison. An extension of the result gives a weaker lower bound on the number of messages if nodes know about their neighbors to distance up to $b, b > 1$.

Yamashita and Kameda [101, 102] proved impossibility results about computation in general graphs in which the nodes are indistinguishable and have partial information about the graphs.

2.5 Communication Protocols

There have been some isolated impossibility results about communication protocols; it seems as if there is much more to be done here.

Aho, Ullman, Weiner and Yannakakis [4] showed that certain kinds of data link behavior cannot be achieved with protocols composes of finite-state machines of particular sizes. The arguments are based on the limitations imposed by small numbers of states. Arguments use case analysis.

Lynch, Mansour and Fekete [78] gave impossibility results for implementing desirable data link behavior (reliable message delivery) in terms of typical physical channel behavior (less reliable packet delivery), in either of two cases: (1) if crashes can occur that cause a loss of memory, or (2) if there are only a bounded number of packet headers for use on the physical channel and a *best case* bounded number of packets are required to deliver each message.

The basic idea of the proofs is that the physical channel can "steal" some packets, while it accomplishes the delivery of messages. This is because the algorithms are supposed to tolerate packet loss. Then the "stolen" packets can be used to fool the receiver process into thinking another message is to be delivered.

These theorems have apparently seemed natural to people in the practical communication protocols community, in fact almost part of the "folk wisdom". Our proofs serve to make these intuitions rigorous. They also make the necessary assumptions explicit, something that network designers might not think about because they take them for granted. For instance, I doubt that a network designer would have realized that a bound on the best case number of packets per message would have been needed for a result such as our second. We as theoreticians are supposed to identify such hidden assumptions.

In fact, although we did not think so at the time, it turns out that this technical-sounding assumption is necessary! For, Attiya, Fischer, Wang and Zuck [11] have recently devised a counterexample algorithm that works with finitely many headers, but does not have such a best-case bound! (At the time we found out about this result, we were trying to prove that such an algorithm was impossible.) Although this algorithm, is very interesting as a counterexample algorithm it not practical, since it uses more and more packets, even in the best case, to send later and later messages. This means the network operation must get slower, and s l o w e r, and s l o w e r ... Some interesting open questions remain about the rate at which the number of packets required must grow with the number of messages delivered.

We found defining the model to be a difficult part of the work in [78]. We used I/O automata; in fact, this was our first attempt to use I/O automata to prove impossibility results. We found getting the formal definitions right to be exceedingly tricky, especially compared to the informal way in which we first discussed the ideas. Much of the difficulty, as usual, involved the proper handling of admissibility. Another difficulty involved modeling the interaction of algorithms; the components about which we prove impossibility results interact with other components, the "physical channels". Thus, constructing a counterexample requires not only giving a bad execution, but also constructing a particular physical channel that interacts with the algorithm to generate the bad execution. Admissibility must also be handled properly for the physical channels.

It is not clear what impact the choice of the I/O automaton model had on the difficulty of this work. My feeling is that the model worked rather well, even though the definitions in [78] are not easy to understand. I think that some of the difficulty is due to the subtlety of the concepts and some due to the fact that our definitions could still use some polishing. But I think the basic model is well suited to expressing all the required concepts.

Spinelli [97] also proved essentially the same impossibility result as the first one in [78], on crash-tolerance. A completely different style of impossibility result about communication protocols appears in [18]; the authors prove a linear lower bound on the amount of time required for deterministic broadcast in a multiaccess medium.

2.6 Miscellaneous

Coan, Kolodner and Oki [32] proved the only example I have of an impossibility result for concurrently-accessible databases. It gives simple proofs of limitations on what types of transactions can execute in a partitioned network. This looks like a good area for future work.

Yao [103] and many others have written a series of papers about the communication complexity of computing particular functions, where the inputs are distributed between several (usually 2) participants. The results are lower bounds on the number of bits that need to be transmitted. The arguments are information-theoretic.

The one example I know involving cryptographic protocols (outside of the authenticated Byzantine agreement work) is the work of Dwork and Stockmeyer [49] giving limitations of the power of interactive proof systems in which the components are finite automata. The limitations are based on the structure of finite-state machines.

Chandy and Misra [29] showed that termination detection requires at least as many messages as the underlying computation whose termination is being detected. They also proved a simple lower bound on the number of messages required for distributed solutions to the dining philosophers problem. The proofs use formal reasoning about knowledge.

Finally, Anderson and Gouda [6] devised a new proof of the impossibility of building an arbiter out of Boolean gates (the "arbiter glitch") problem. Their proof is based on discrete bivalence considerations rather than continuous considerations such as the other proofs in the literature. They make a restrictive assumption that there not be any gates with outgoing wires connected back to inputs of the same gate (even with delay on the wire). It would be interesting to understand what happens if this restriction is removed.

3 General Comments

So what can be distilled from this survey?

3.1 The Basic Ideas That Make The Proofs Work

There is only one fundamental underlying idea on which all of the proofs in this area are based, and that is the *limitation imposed by local knowledge in a distributed system.* If a process sees the same thing in two executions, it will behave the same in both. Ideas related to local knowledge have been used implicitly in proofs since the beginning, although in the past few years there has been some work in trying to make the use of knowledge explicit.

There are many reasons for the limits on local knowledge in distributed settings. Uncertainty arises from asynchrony, failures, and unknown inputs. Information about other parts of the system might not be communicated quickly because of limitations on communication media, e.g., the size of shared memory, the bandwidth of message channels, or the distance information must travel.

Many specific techniques are used, all manifestations of the limitation of local knowledge. I have mentioned ***pigeonhole arguments*** for bounds on the number of values of shared memory, ***scenario arguments*** for bounds on the number of processes, ***chain arguments***, primarily for lower bounds on rounds for consensus problems, ***bivalence arguments*** for impossibility of decision problems, ***communication diagram stretching arguments*** for time and message bounds for synchronization problems, ***symmetry arguments*** for impossibility and message lower bounds for network computations, especially for ring computations, ***distance arguments*** for message bounds in low-degree networks such as rings, ***crossing sequence arguments*** for ring computations, ***message-stealing arguments*** for communication protocols, and ***finite-state arguments*** for FSA-based algorithms.

Also, some proofs make use of reducibilities to infer impossibility results from others that have previously been proved.

3.2 Connections with Formal Modeling

The work of doing impossibility proofs is tightly intertwined with the work of defining formal models.

First, impossibility proofs need to be based on rigorous and well-designed formal models. It may be possible to avoid using formal models if one is interested only in designing algorithms. But it is not possible to fake an impossibility proof - such a proof makes no sense at all without rigorous description. That is not to say that one shouldn't work on an impossibility proof at an informal level; the final product, however, needs to be carefully described.

There are many features that make a model appropriate for impossiblity proofs. Of course, it needs to be rigorous. It must permit separate descriptions of the problems to be solved and of the allowable implementations. It must provide a proper treatment of admissibility and control of actions. Problem statements must be sufficiently "tight" to serve as a reasonable contract between a specifier and an implementor. (They should neither say too little nor too much.) Problem statements must be sufficiently clean to be invoked repeatedly as justifications for steps of a construction. Finally, implementation models need to be clean and simple. (It is neither feasible nor interesting to prove impossibility results about a messy implementation model.)

A by-product of work on impossibility proofs is the development of formal models with the nice features listed above. If an area has only algorithms, but no impossibility results, I don't believe it is likely that the models that arise are likely to have the same features. In particular, the problem statements are not likely to be either tight or clean.

When many people get involved in proving upper and lower bound results in an area, the problem statements and implementation assumptions used in that area tend to get a lot of careful discussion, which in turn helps lead to convergence on good sets of assumptions.

The use of formal models forces people to make their assumptions explicit. This helps to expose subtle differences in assumptions, which often leads to many variations on the same problem, with corresponding different results.

On the negative side, it is certainly true that the use of rigorous formal models imposes overhead on the presentation of results; for impossibility results, I think this is unavoidable.

3.3 Problem Statements

I have some general remarks about appropriate kinds of problem statements for impossibility results.

First, the problems must be stated precisely. This does not mean that they have to be stated in a formal language such as temporal logic. It does mean that they must make sense in terms of a basic mathematical model that can be used for describing implementations and for carrying out the necessary mathematical arguments.

It is not enough for the problem statements to be precise; the problems also need to be well-chosen - crisp and simple. This makes it a lot easier to invoke the problem statements when carrying out constructions of bad executions. It also makes it more likely that the results obtained will be fundamental.

It is very hard to work on a direct impossibility proof for solving a very complex distributed computing problem, e.g., for implementing a fancy distributed UNIX in the presence of certain faults of the implementing processors, perhaps with a certain cost. One needs to extract simple prototype problems from an area to carry out the basic proofs. (Sometimes complex and specialized problems can be shown to be impossible using reducibilities, or by being special cases of a result about simple, more general systems.)

It is also important for problem statements to be as general as possible, although generality seldom comes on the first try.

In a paper with contrasting possibility and impossibility results, it is not unusual to find two different statements for the "same" problem - a strong statement for the algorithms and a weak statement for the impossibility results. This strategy is a way of making each result as strong as possible.

3.4 The Process of Working on Such Proofs

How does one go about working on an impossibility proof? The first thing to do is to try to avoid solving the problem, by using a reducibility to reduce some other unsolvable problem to it. If this fails, you next consider your intuitions about the problem. This might not help much either: in my experience, my intuitions about which way the result will go have been wrong about 50% of the time.

Then it is time to begin the game of playing the positive and negative directions of a proof against each other. My colleagues and I have often worked alternately on one direction and the other, in each case until we got stuck. It is not a good idea to work just on an impossibility result, because there is always the unfortunate possibility that the task you are trying to prove is impossible is in fact possible, and some algorithm may surface.

An interesting interplay often arises when you work alternately on both directions. The limitations you find in designing an algorithm - e.g., the reason a particular algorithm fails - may be generalizable to give a limitation on all algorithms. This is how we found the lower bound in [56]. Conversely, the reasons that an impossibility proof fails can sometimes be exploited to devise counterexample algorithms. This is how we found the no lockout algorithm in [26].

Arriving at a careful statement of the problem is usually an iterative process. It usually takes a while just to get it correct: it's easy to make the problem statement too strong (e.g., by requiring that a resource be granted without saying that the environment must always return the resource), in which case impossibility results might hold for trivial reasons. It's also easy to make the statement too weak, in which case trivial counterexample algorithms can arise.

With some luck, this iterative process eventually leads to an interesting problem statement and a corresponding impossibility result; then the problem statement should be "polished". Assumptions that are not needed can be eliminated, so that impossibility is proved based on the weakest possible set of requirements. The problem statement should be made as general and elegant as possible. (For example, in the impossibility result of [55], we weakened the usual consensus validity conditions after the fact to include any algorithm with the *option* of reaching either of two different decisions. This meant that the result was strong enough to apply to commit algorithms. We also noticed after the fact that we could strengthen the power of the message system from individual sends to atomic broadcast; this strengthening weakens the requirements of the algorithm, since it now is only required to work in a stronger environment.)

I find that one of the hardest aspects of working out problem statements and impossibility proofs (especially for asynchronous systems) is the proper treatment of admissibility. The definitions and proofs must ensure that all (non-failed) processes continue to take steps, or all messages are delivered, or that other appropriate liveness conditions are satisfied, in the bad admissible executions that are constructed.

3.5 What Good Are Impossibility Results?

What good are impossibility results, anyway? They don't seem very useful at first, since they don't allow computers to do anything they couldn't previously.

Most obviously, impossibility results tell you when you should stop trying to devise or improve an algorithm. This information can be useful both for theoretical research and for systems development work.

It is probably true that most systems developers, even when confronted with the proved impossibility of what they're trying to do, will still keep trying to do it. This doesn't necessarily mean that they are obstinate, but rather that they have some flexibility in their goals. E.g., if they can't accomplish something absolutely, maybe they can settle for a solution that works with "sufficiently high probability". In such a case, the effect of the impossibility result might be to make a systems developer clarify his/her claims about what the system accomplishes.

Proving impossibility results causes us to take a very analytical approach to understanding the area. It causes us to state carefully exactly what assumptions (about the execution environment and the problems) the results depend on. This sort of detailed information does not normally arise from or algorithm or system development work alone.

Sometimes impossibility proofs lead to interesting work on ways of getting around the inherent limitation. For example, many randomized algorithms have been produced in order to get around the inherent cost previously proved for deterministic and nondeterministic algorithms. Examples include Ben-Or's asynchronous fault-tolerant consensus algorithm in [19], Itai and Rodeh's randomized algorithms for leader election without identifiers in [66] and Feldman and Micali's fast algorithm for synchronous consensus [52].

The close connections between impossibility proofs and modeling means that impossibility results help in the development of formal models. Models produced for impossibility proofs have many nice features, as I discussed earlier. They are not only useful for proving impossibility results; they also have other uses, such as specification and verification of algorithms and software.

Finally, I think that an understanding of impossibility results in an area is an important part of understanding the fundamental ideas of that area.

3.6 Unified Models

A pet question of mine is what we can do to reduce the need for so much definitional and modeling work for impossibility results. Those of us who prove impossibility results get tired of writing those long and formalism-laden definitions sections, and I am sure most people are tired of reading them. Since such precision is necessary, it seems that the only hope is to try to avoid repeated work by using a standard model as the foundation. I am not sure yet how successful that will be.

A unified model could provide a standard way of coping with ideas that appear repeatedly. For example, the I/O automaton model provides more-or-less standardized ways of presenting algorithms and problem statements (for an asynchronous setting), and has a built-in treatment of admissibility and of control of actions. These considerations arise in many different results, in many areas, including shared memory algorithms, distributed consensus and network algorithms.

Use of a unified model that spans several areas could facilitate the application of results from one area to another area, e.g., the application of consensus results to mutual exclusion or register problems. (This is true not just for impossibility results.)

I have tried using both of the general models I have been involved with, in impossibility proofs. The model of [81] was not that successful for this purpose, but in retrospect I think it was mainly because it is a shared-memory model and we were trying to use it for inappropriate settings such as message-passing systems. The I/O automaton model has been used recently, and seems reasonably successful.

I have considered recasting some earlier results in terms of I/O automata. In the early work on mutual exclusion, the definitions did not establish a clean boundary around the algorithms, allowing their interactive behavior to be clearly specified at that boundary and making it possible to compose the algorithms with others to build a system. I/O automata could remedy this. On the other hand, I/O automata have one drawback for this area: the fact that they treat inputs and outputs as separate events means that they might tend to treat some things non-atomically that could be treated atomically. This could complicate the proofs.

I think that impossibility results about atomic registers could expressed well using I/O automata. For consensus, the result of [55] and other related results can also be redone using I/O automata; the new presentations seem to me to be a little simpler than the old. The synchronization result of [8] can also be redone using I/O automata rather than our shared-memory model, and the new presentation seems much simpler and more natural than the old.

I don't expect a unified model to be a panacea. There are many ideas that are *not* common to all work in the area, such as special assumptions about timing and failures. Each result would probably still need to be preceded by a description of its own set of special conditions. But perhaps these might constitute less overhead than before. Perhaps the use of a general model might help to identify which of the differences are essential, and remove the others.

It does not actually seem that thinking about a general model such as the I/O automaton model has yet been very helpful in getting insight while working on the combinatorial results. So far, their use has been solely in producing clear and rigorous presentations (and finding mistakes in intuitions).

3.7 Randomized Algorithms

So far, there have been very few interesting impossibility results for randomized algorithms. The main examples I have mentioned are [68, 60, 33, 1]. Of course, one would expect fewer impossibility results for randomized algorithms, because less is impossible with such algorithms, but some more should be provable than exist currently.

It is much harder to reason about the limitations of randomized algorithms than about those of deterministic algorithms; it seems necessary to analyze very complex probabilistic interactions between the algorithms and adversaries having various amounts of knowledge and power. The area of adversarial computing is one that really could use improved understanding, and impossibility results for randomized algorithms would surely contribute to that understanding.

3.8 They're Easy

Impossibility proofs are much easier in our area than in most others. This is because the limitation of local knowledge is the fundamental fact about the setting in which we work, and it is a very powerful limitation.

4 Future Directions

4.1 Technical Open Questions

I mentioned a few open questions earlier. These are summarized here.

1. In the no-lockout mutual exclusion work in [26], is the "forgetting" assumption necessary?

2. In the consensus work in [46], where some assumptions are made about time for message delivery, what are the exact time bounds required for consensus?

3. With what probability can consensus be guaranteed by randomized algorithms, in the presence of a large number t of faults relative to the total number n of processes?

4. What is the exact number of process names required by the process renaming problem of [10]; is it $n + 1$ or $n + t$ or somewhere in between?

5. In the data link work of [78], how fast must the number of packets grow with time? (Some new results appear in [99], in the current PODC.)

6. More results in the style of [65] should be possible. Exactly what objects can and can't be implemented in terms of what other objects, in a wait-free manner, or not in a wait-free manner? What are the associated time bounds?

7. Can the result in [6] be extended to the case in which the circuit does not have a loop-free restriction?

4.2 Other Areas

Impossibility results in distributed computing theory have been concentrated into a few subareas. It should be possible to expand the set of problems being considered, by looking at other areas. There will be some initial work required to identify crisp problems suitable for impossibility results.

Although some of these areas have already been well "mined" for basic algorithms, the same is not true for impossibility results (and counterexample algorithms). Some results could arise based on the folk wisdom of the areas. Some suggestions for areas are:

1. Communication protocols: Not very much has been done yet. There is still more to understand about the relationship between the data link layer and the physical layer, and there are lots of other layers to consider.

2. Real-time processing: It would be nice to have a theory to describe the fundamental combinatorial properties of real-time systems. Impossibility results should be an important part of this. Note that both this and the preceding area require models for timing-dependent algorithms.

3. Parallel computing: There has been lots of combinatorial work in this area, but the models (PRAM's, etc.) are different from those commonly used in our area. We might want to consider models for parallel computing that are similar to the models that have been considered in our area - involving asynchrony and failures, for example.

4. Databases: Little has been done so far. This area is characterized by complex problems and algorithms; it is necessary to identify simple, crisp problems. It might be possible to prove limitations on the ability of systems to guarantee serializability with liveness, e.g., based on limited information provided to each object, or based on kinds of faulty behavior. Results might be obtained about specific data types or transaction types.

4.3 Other Styles of Results

I would like to see more lower bounds on time for asynchronous algorithms, such as [8]. Such bounds have been underemphasized so far. Time measure definitions appropriate for asynchronous systems, such as those in [90, 81, 79, 83] must be used. More work is also needed on impossibility results for randomized algorithms. More work is needed on concepts relating different problems, such as reducibilities and complexity or computability classes. (Such classes have been very useful elsewhere in complexity theory.)

4.4 Modeling

More work is needed in developing good models for use in proving impossibility results for distributed computing. A general model is desirable; I/O automata are one possibility, but there may be others. If I/O automata are to be used, they need to be augmented in various ways, e.g., with time definitions as in [83].. It will still sometimes be necessary to develop models tailored for specific areas. Perhaps a general model can be used, with special structure added on to fit it to each area.

4.5 Unifying and Generalizing Results

It may be useful to try to unify the work that's already been done, in the way that [54] unified a large collection of $n \leq 3t$ lower bounds. In particular, the results about ring computation could use such coalescing. There seem to be too many $\Omega(n \log n)$ lower bounds!

There seems to be something very similar about the problems of mutual exclusion, consensus, serializability, leader election, and even global snapshots. So there should be similar inherent limitations on solving these problems. Are there common proof techniques, or even reducibilities here?

Although there are 100 proofs, maybe there are only six ideas - perhaps it is possible to prove the Six Fundamental Theorems of Distributed Computing, from which all of these other results will follow!

5 Conclusions

I've tried in this talk to give you a good picture of the history, status and flavor of research in impossibility proofs for distributed computing. I hope you're convinced that it is an interesting and fruitful area for research. Now with some luck, skill and inspiration, we can continue to make great strides, proving more and more things to be impossible!

Acknowledgements: I would like to thank Hagit Attiya, Liuba Shrira and Mark Tuttle, who corrected many inaccuracies and tracked down several results.

References

[1] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. *Probabilistic Solitude Detection II: Ring Size Known Exactly.* Technical Report 87-11, University of British Columbia, Vancouver, B.C., Canada, April 1987.

[2] Hosame Abu-Amara. *Fault-tolerant Distributed Algorithms for agreeement and election.* Phd Thesis UILU-ENG-88-2242, ACT-95, University of Illinois at Urbana-Champaign, August 1988.

[3] Y. Afek and E. Gafni. Time and message bounds of election in synchronous and asynchronous complete networks. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*, pages 186–195, Minaki, Ontario, Canada, August 1985.

[4] A. V. Aho, J. D. Ullman, A. D. Wyner, and M. Yannakakis. Bounds on the size and transmission rate of communication protocols. *Computers and Mathematics with Applications*, 8(3):205–214, 1982. This is a later version of [5].

[5] A. V. Aho, J. D. Ullman, and M. Yannakakis. Modeling communication protocols by automata. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 267–273, 1979.

[6] J. Anderson and M. Gouda. *A new explanation of the Glitch Phenomenon.* Technical Report TR-88-23, Department of Computer Science, The University of Texas at Austin, Austin, TX 78712-1188, June, 1988.

[7] D. Angluin. Local and global properties in networks of processors. In *Proceedings of 12th STOC*, pages 82–93, 1980.

[8] E. Arjomandi, M. Fischer, and N. Lynch. Efficiency of synchronous versus asynchronous distributed systems. *J. ACM*, 30(3):449–456, July 1983.

[9] A. Attiya, D. Dolev, and J. Gil. Asynchronous Byzantine consensus. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 119–133, Vancouver, B.C., Canada, August 1984.

[10] H. Attiya, A. Bar-Noy, D. Dolev, D. Koller, D. Peleg, and R. Reischuk. Achievable cases in an asynchronous environment. In *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, pages 337–346, October 1987.

[11] H. Attiya, M. Fischer, D. Wang, and L. Zuck. Reliable communication over an unreliable channel. In progress.

[12] H. Attiya and Y. Mansour. Language complexity on the synchronous anonymous ring. *Theoretical computaer science*, 53(3):167–185, 1987.

[13] H. Attiya and M. Snir. Better computing on an anonymous ring. Submitted to publication.

[14] Hagit Attiya, Marc Snir, and Manfred K. Warmuth. Computing on an anonymous ring. October 1988.

[15] B. Awerbuch, O. Goldreich, D. Peleg, and R. Vainish. A tradeoff between information and communication in broadcast protocols. In *3rd Aegean workship on theory of computing*, pages 369–379, Springer-Verlag, 1988.

[16] Baruch Awerbuch. Communication-time tradeoffs in network synchronization. In *Proceedings of the 4th Annual ACM Symposium on Principles of Distributed Computing*, pages 272–276, ACM, August 1985.

[17] O. Babaoglu, P. Stephenson, and R. Drummond. Reliable broadcasts and communication models: tradeoffs and lower bounds. *Distributed Computing*, 2:177–189, 1988.

[18] R. Bar-Yehuda, O. Goldreich, and A. Itai. On the time-complexity of broadcast in radio networks: an exponential gap between determinism and randomization. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, British Columbia, Canada, August 1987.

[19] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proceedings of 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, August 1983.

[20] Ofer Biran, Shlomo Moran, and Shmuel Zaks. A combinatorial characterization of the distributed tasks which are solvable in the presence of one faulty processor. In *Proceedings of the 7th Annual ACM Symposium on Principles*

of Distributed Computing, pages 263–275, August 1988.

[21] B. Bloom. Constructing two-writer atomic registers. In *Proceedings of 6th ACM Symposium on Principles of Distributed Computing*, pages 249–259, Vancouver, British Columbia, Canada, August 1987. Also, to appear in special issue *IEEE Transactions On Computers*.

[22] H.L. Bodlaender. Distributed algorithms, structure and complexity. 1986. Ph.D. Thesis.

[23] L. Bouge. On the existence of symmetric algorithms to find leaders in networks of communication sequential processes. Rept. No. 86-18, LITP, Univ. Paris 7, Paris(1986). To appear in Acta Informatica.

[24] M. Bridgeland and R. Watro. Fault tolerant decision making in totally asynchronous distributed systems. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 52–63, August 1987.

[25] J. Burns. *A formal model for message passing systems*. Technical Report TR-91, Computer Science Dept., Indiana University, May 1980.

[26] J. Burns, M. Fischer, P. Jackson, N. Lynch, and G. Peterson. Data requirements for implementation of n-process mutual exclusion using a single shared variable. *Journal of the ACM*, 29(1):183–205, 1982.

[27] J. Burns and N. Lynch. Mutual exclusion using indivisible reads and writes. In *Proceedings of 18th Annual Allerton Conference on Communications, Control, and Computing*, pages 833–842, 1980. Revision in progress.

[28] K. M. Chandy and J. Misra. On the nonexistence of robust commit protocol. January 1987. Manuscript.

[29] K. Mani Chandy and Jayadev Misra. How processes learn. *Distributed Computing*, 1(1):40–52, 1986.

[30] Benny Chor, Amos Israeli, and Ming Li. On processor coordination using asynchronous hardware. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 86–97, Vancouver, British Columbia, Canada, August 1987.

[31] B. Coan, D. Dolev, C. Dwork, and L. Stockmeyer. The distributed firing squad problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, Rhode Island*, pages 335–345, May 1985.

[32] B. Coan, B. Oki, and E. Kolodner. Limitations on database availability when networks partition. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 187–194, Calgary, Alberta, Canada, August 1986.

[33] B. Coan and J. Welch. Transaction commit in a realistic fault model. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 40–51, Calgary, Alberta, Canada, August 1986.

[34] B.A. Coan. *Achieving Consensus in Fault-Tolerant Distributed Computer Systems: Protocols, Lower Bounds, and Simulations*. PhD thesis, Massachusetts Institute Technology, June 1987.

[35] A.B. Cremers and T.N. Hibbard. An algebraic approach to concurrent programming control and related complexity problems. *Symposium on Algorithms and Complexity*, April 1976.

[36] Dolev D., Lynch N., Pinter S., Stark E., and Weihl W. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):449–516, 1986.

[37] R. DeMillo, N. Lynch, and M. Merritt. Cryptographic protocols. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, San Francisco, California*, pages 383–400, May 1982.

[38] E.W. Dijkstra. Solution of a problem in concurrent programming control. *Communications Of The ACM*, 8(9):569, September 1965.

[39] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3:14–30, 1982.

[40] D. Dolev and C. Dwork. A study of communication primitives. 1989. Unpublished Manuscript.

[41] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97, 1987.

[42] D. Dolev and R. Reischuk. Bounds on information exchange for Byzantine agreement. In *Proceeding of ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, pages 132–140, Attawa, Canada, August 1982.

[43] D. Dolev and H.R. Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Computing*, 12(4):656–666, November 1983.

[44] S. Dolev, J. Halpern, and R. Strong. On the possiblity and impossibility of achieving clock synchronization. In *Proceedings of 16th Symposium on Theory of Computing*, pages 504–510, May 1984. Journal of Computer and System Sciences, 32:230–250, 1986.

[45] P. Duris and Z. Galil. Two lower bounds in asychronous distributed computation. *28th Annual Symposium on Foundations of Computer Science*, 326–330, October 1987.

[46] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[47] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine Environment I: crash failures. In *Proceedings of Conference on Theoretical Aspects of Reasoning about Knowledge*, 1986. Submitted to *Information and Computation.*

[48] C. Dwork and D. Skeen. The inherent cost of nonblocking commitment. In *Proceedings of the 2nd Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada*, pages 1–11, August 1983.

[49] Cynthia Dwork and Larry Stockmeyer. *Interactive Proof Systems with Finite State Verifiers.* Research Report RJ 6262, IBM Almaden Research Center, may 1988.

[50] A. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, pages 73–87, Calgary, Alberta, Canada, August 1986. Revised and submitted for Publication.

[51] A. Fekete. Asynchronous approximate agreement. In *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing*, pages 64–76, August 1987.

[52] Paul Feldman and Silvo Micali. Optimal algorithms for Byzantine agreement. May 1988. Paul Feldman Phd thesis.

[53] M. Fischer, N. Lynch, J. Burns, and A. Borodin. Distributed FIFO allocation of identical resources using small shared space. *ACM Transactions on Programming Languages and Systems*, 11(1):90–114, January 1989.

[54] M. Fischer, N. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1:26–39, 1986.

[55] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of 2nd ACM Symposium on Principles of Database Systems*, pages 1–7, Atlanta, GA, March 1983. Also appears as Technical Report MIT/LCS/TR-282, Massachusetts Institute Technology, Laboratory for Computer Science, Cambridge, MA 02139, September 1982. Revised version in *J. ACM*, 32(2):374-382, April 1985.

[56] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.

[57] Michael J. Fischer, Nancy A. Lynch, James E. Burns, and Allan Borodin. Resource allocation with immunity to limited process failure. In *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, pages 234–254, October 1979.

[58] G.N. Frederickson and N. Lynch. Electing a leader in a synchronous ring. *J. ACM*, 34(1):98–115, January 1987.

[59] O. Goldreich and L. Shrira. The effects of link failures on computation in asynchronous rings. In *Proceedings of the 5th Annual ACM Symposium on Principles of Distributed Computing*, Calgary, Alberta, Canada, August 1986.

[60] R.L. Graham and A.C. Yao. On the improbability of reaching Byzantine agreement. In *Proceedings of 21st ACM Symposium on Theory of Computing*, May 1989.

[61] J. Gray. *Notes on Data Base Operating Systems.* Technical Report IBM Report RJ2183(30001), IBM, February 1978. (Also in Operating Systems: An Advanced Course, Springer-Verlag Lecture Notes in Computer Science #60.).

[62] Vassos Hadzilacos. A knowledge-theoretic analysis of atomic commitment protocols. In *Proceedings of the 6th Annual ACM Symposium on Principles of Database Systems*, 1987. Revised version available, submitted for publication.

[63] J. Halpern, N. Megiddo, and A. Munshi. Optimal precision in the presence of uncertainty. *Journal of Complexity*, 1(2):170–196, December 1985.

[64] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. In *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, pages 50–61, 1984. To appear in JACM. A revised version appears as *IBM Research Report RJ 4421*, Third Revision, September, 1988.

[65] Maurice Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the 7th Annual ACM Symposium on Priciples Distributed Computing*, pages 276–290, Toronto, Ontario, Canada, August 1988.

[66] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. In *Proceedings of 22nd Annual ACM Symposium on Foundations of Computer Science*, pages 150–158, 1981.

[67] Ralph E. Johnson. *Symmetry in Distributed Systems. Phd Thesis*. Technical Report TR-87-855, Department of Computer Sciene, Cornell University, Ithaca, New York, 14853-7501, August 1987.

[68] A.R. Karlin and A.C. Yao. Lower bounds to randomized for the Byzantine generals problem. 1984. unpublished.

[69] R. Koo and S. Toueng. Effects of message loss on the termination of distributed protocols. *Information Processing Letters*, 27:181–188, April 1988.

[70] E. Korach, S. Moran, and S. Zaks. Tight lower and upper bounds for some distributed algorithms for a complete network of processors. In *In Proceedings of 3rd ACM Symposium on Principles of Distributed Computing*, pages 199–207, 1984.

[71] L. Lamport. On interprocess communication. *Distributed Computing*, 1(1):77–101, 1986.

[72] L. Lamport. The weak Byzantine generals problem. *Journal of the ACM*, 30(3):669–676, 1983.

[73] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[74] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, July 1978.

[75] Leslie Lamport and P. M. Melliar-Smith. Byzantine clock synchronization. In Jayadev Misra, editor, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pages 68–74, Association for Computing Machinery, Inc., New York, August 1984.

[76] M. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1987.

[77] J. Lundelius and N. Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2-3):190–204, August/September 1984.

[78] N. Lynch, Y. Mansour, and A. Fekete. The data link layer: two impossibility results. In *Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing*, pages 149–170, Toronto, Canada, August 1988. Also, Technical Memo MIT/LCS/TM-355, Lab for Computer Science, Massachusetts Institute Technology, Cambridge, MA, May 1988.

[79] N. Lynch and M.R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the 6th Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987.

[80] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. To be published in *Centrum voor Wiskunde en Informatica Quarterly*. Also in Technical Memo, MIT/LCS/TM-373, Lab for Computer Science Massachusettes Institute of Technology, November 1988.

[81] Nancy A. Lynch and Michael J. Fischer. On describing the behavior and implementation of distributed systems. *Theoretical Computer Science*, 13(1):17–43, January 1981.

[82] Yishay Mansour and Shmuel Zaks. On the bit complexity of distributed computations in a ring with a leader. *Information and Computation*, 75(2):162–177, 1987.

[83] F. Modugno, M. Merritt, and M.R. Tuttle. Time constrained automata. November 1988. Unpublished manuscript.

[84] S. Moran and M. Warmuth. Gap theorems for distributed computing. In *Proceedings of the* 5^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 131–150, August 1986.

[85] S. Moran and Y Wolfstahl. Extended impossibility results for asynchronous complete networks. *Information Processing Letters*, 26:145–151, 1987.

[86] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[87] J. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum-finding algorithms. *Journal of ACM*, 31(4):905–919, October 1984.

[88] Jan K. Pachl. A lower bound for probabilistic distributed algorithms. *Journal of Algorithms*, 8(1):53–65, March 1987.

[89] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[90] G.L. Peterson and M.J. Fischer. Economical solutions for the critical section problem in a distributed syste. In *Proceedings of 9th STOC*, pages 91–97, May 1977.

[91] M. Rabin. Randomized Byzantine generals. In *Proceedings of 24th Symposium on Foundations of Computer Science*, pages 403–409, November 1983.

[92] M. O. Rabin. The choice coordination problem. *Acta Informatica*, 17(2):121–134, 1982.

[93] R. Reischuk and M. Koshors. Lower bound for synchronous systems and the advantage of local information. In *Proceedings of th* 2^{nd} *International Workshop on Distributed Algorithms*, Amsterdam, June 1987.

[94] Nicola Santoro. *On the message complexity of Distributed Problems.* Technical Report SCS-TR-13, School of Computer Science, Carleton Univerity, Ottawa, Canada, December 1982.

[95] R. Schaffer. On the correctness of atomic multi-writer registers. Bachelor's Thesis, June 1988, Massachusetts Institute Technology. Also, Technical Memo MIT/LCS/TM-364.

[96] A. Segall and O Wolfson. Transaction commitment at mimimal communication cost. In *Proceedings of the* 6^{th} *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 112–117, San Diego, California, March 1987.

[97] John Spinelli. *Reliable Data Communication in Faulty Networks.* Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, 1988.

[98] G. Taubenfeld, S. Katz, and S. Moran. Impossibility results in the presence of multiple faulty processes. April 1988. Submitted for publication.

[99] D. Wang and L. Zuck. Tight bound for the sequence transmission problem. To appear in PODC 89.

[100] Jennifer L. Welch. Simulating synchronous processors. *Information and Computation*, 74(2):159–171, 1987.

[101] M. Yamashita and T. Kameda. *Computing on an Anonymous Network.* Technical Report LCCR-TR-87-15, Simon Fraser University, Burnaby, British Columbia, 1987.

[102] M. Yamashita and T. Kameda. Computing on an anonymous network. In *Proceedings of the* 7^{th} *Annual ACM Symposium on Priciples Distributed Computing*, pages 131–148, Toronto, Ontario, Canada, August 1988.

[103] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the* 11^{th} *Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.