RANDOMIZED DISTRIBUTED COMPUTING ON RINGS

by

Lisa Higham

Technical Report 89-5

Abstract

The communication complexity of fundamental problems in distributed computing on an asynchronous ring are examined from both the algorithmic and lower bound perspective. A detailed study is made of the effect on complexity of a number of assumptions about the algorithms. Randomization is shown to influence both the computability and complexity of several problems. Communication complexity is also shown to exhibit varying degrees of sensitivity to additional parameters including admissibility of error, kind of error, knowledge of ring size, termination requirements, and the existence of identifiers.

A unified collection of formal models of distributed computation on asynchronous rings is developed which captures the essential characteristics of a spectrum of distributed algorithms — those that are error free (deterministic, Las Vegas, and nondeterministic), and those that err with small probability (Monte Carlo and nondeterministic/probabilistic). The nondeterministic and nondeterministic/probabilistic models are introduced as natural generalizations of the Las Vegas and Monte Carlo models respectively, and prove useful in deriving lower bounds. The unification helps to clarify the essential differences between the progressively more general notions of a distributed algorithm. In addition, the models reveal the sensitivity of various problems to the parameters listed above.

Complexity bounds derived using these models typically vary depending on the type of algorithm being investigated. The lower bounds are complemented by algorithms with matching complexity while frequently the lower bounds hold on even more powerful models than those required by the algorithms.

Among the algorithms and lower bounds presented are two specific results which stand out because of their relative significance.

1. If g is any nonconstant cyclic function of n variables, then any nondeterministic algorithm for computing g on an anonymous ring of size n has complexity $\Omega(n\sqrt{\log n})$ bits of communication; and, there is a is nonconstant cyclic boolean function f, such that f can be computed by a Las Vegas algorithm in $O(n\sqrt{\log n})$ expected bits of communication on a ring of size n.

2. The expected complexity of computing AND (and a number of other natural functions) on a ring of fixed size n in the Monte Carlo model is $\Theta(n \min\{\log n, \log\log(1/\epsilon)\})$ messages and bits where ϵ is the allowable probability of error.

Contents

Acknowledgements

It has been my very good fortune to have studied under the supervision of David Kirkpatrick. I am deeply indebted to him for the example he has set as a supervisor, a scientist and a human being. I cherish the opportunity I have had to work with him and his example is one I shall always strive to emulate.

Andrew Adler has participated in all aspects of my research to a far greater extent than the normal involvement of a committee member. I particularly appreciate his repeated guidance through techniques of probability theory, insisting that I "get it right". Sam Chanson and Alan Wagner, the remaining members of my committee, contributed valuable suggestions and brought fresh alternative perspectives to my work.

Karl Abrahamson has been more than a very active collaborator over the past four years. Rather, he has played a role akin to that of an unofficial second supervisor. I am grateful to him for his invaluable insights, as well as for his guidance and encouragement.

My appreciation also goes to my external examiner, Nancy Lynch, for her thorough reading of a draft of this dissertation. Her perceptive comments led to several improvements in the final version.

In spite of my endless demands on his ear, Alan Covington has responded with an untiring willingness to listen and with critical and helpful suggestions. This is in addition to a bottomless well of support, patience, and love for which I am very thankful.

I suspect that one rarely has such a richly rewarding experience as I have enjoyed as a graduate student. This happy circumstance is due not only to those mentioned above but also to the community of fellow graduate students and department members. To all of the large support group of friends both inside and outside the department of Computer Science at The University of British Columbia go my heartfelt thanks.

Chapter 1

Introduction

A loosely connected network of small processors is a common alternative to a large centralized resource. The processors in such a network cooperate to solve a problem by exchanging messages. Typically, each processor executes a set of local instructions which cycle through the steps: i) possibly send a message, ii) wait until a message is received, and iii) do some local processing. The system is programmed by specifying a sequence of these steps for each processor.

The network may consist of many types of processors and the communication links may be implemented using various technologies. Such a system may well be unreliable, but reliable message transfer is typically provided by a protocol that retransmits lost messages after a suitable time period has elapsed. Reliability is thus achieved at the expense of a processor's ability to accurately predict when a message will arrive. The speed with which a message is transmitted and processed is dependent upon the specific processors and transmission lines involved and on the protocol providing reliability. Under these conditions, a processor could not with certainty distinguish a late message from the absence of a message. A conservative approach would then require that algorithms do not assume anything about message delays. There is neither a global clock nor shared memory to force the processors to compute in lock-step, so, apart from the exchange of messages, each processor proceeds independently. To the extent that synchronization is necessary,

it must be achieved by the exchange of messages. Typically, the information required to solve a problem is distributed throughout the network, and processors must combine local computation with the acquisition of additional information until the desired solution is achieved. Communication costs tend to dominate local processing costs. For example, in a long-haul network, each packet of communication may be assessed an actual charge. Furthermore, the time required to package and deliver a message is frequently significantly longer than the time required for local computation. Therefore, a principal motive is to design programs that minimize communication. The total number of messages or bits used by a program serves as a first approximation to its cost.

A system with these characteristics is called an *asynchronous distributed system*. The absence of, or lack of reliance upon, a global clock or any time out mechanism accounts for the "asynchronous" label. It is assumed that messages are subject to unbounded delay, but that those sent over the same communication link are eventually delivered and always in the order in which they were sent. (A large body of research, but not this dissertation, addresses the topic of error-tolerant distributed computing which drops the assumption of reliable message delivery.) It is usual to model a distributed system as a graph with processors represented by vertices and the communication channels represented by edges. An algorithm for a distributed system is usually thought of as a program, one copy of which resides at each processor. The algorithm is executed when each processor simultaneously runs the program. In order to model more general kinds of algorithms, this description is relaxed, in this dissertation, by permitting processors to run different programs. An algorithm for a distributed system is therefore modelled as a strategy for assigning programs to processors.

A simple complexity measure reflects the assumption that communication costs significantly dominate local processing costs. The complexity of a computation is defined to be the total amount of communication (either messages or bits) that transpires during the computation. No charge is attributed to local computation.

The *function evaluation* problem assumes that each processor has an initial input and is required to determine the value of some function of the collection of inputs. Function

evaluation is used to model information sharing by processors.

The distributed environment differs significantly from both the classical single processor environment modelled by a random-access machine (RAM) and from the tightly coupled parallel machine environment modelled by a parallel random-access machine (PRAM). The computation of a deterministic algorithm running on a sequential or parallel machine is completely determined by the input. In contrast, on an asynchronous distributed network, an algorithm with a fixed input may give rise to a number of different computations depending on message delays. In the PRAM model processors are automatically synchronized by a global clock; in the RAM model there is no synchronization to be done. In the asynchronous distributed model, however, synchronization of processors must be maintained explicitly by the algorithm. In both the RAM and PRAM models, processors have access to all information because there is one global memory, whereas, information sharing is necessitated in the distributed model because there are only local memories. Because of these properties, even simple problems present surprisingly subtle difficulties in the distributed environment. For example, deadlock (termination of the computation without arriving at a solution) or livelock (continued computation without progress toward a solution) are possible consequences of failure to ensure processor coordination. Also, the outcome of a computation on an asynchronous network is required to be independent of the pattern of delays during the computation. Hence, the design of correct distributed algorithms and proofs of their correctness can be involved tasks.

In addition to new issues of correctness arising from the pitfalls of asynchrony and concurrency, there are new complexity concerns. Algorithms are measured against the criterion of communication efficiency. Since communication behaves differently than the more familiar resources of time and space, transfer of techniques from the more traditional domains to distributed computing systems is not automatic.

Current distributed systems do not always function as specified. This is evidence that there is a lack of expertise in designing and analysing distributive programs. A standard tactic when studying a complicated problem is to restrict the problem domain by making simplifying assumptions. An asynchronous ring is one of the simplest of network topolo-

gies. This topology eliminates some of the complicating factors encountered in arbitrary networks. On a unidirectional ring, a processor receives communication on only one channel, and thus a scheduler cannot manipulate the order of arriving messages. Even in the bidirectional case the possible computation sequences are under some control. Nonetheless, features uncovered in the ring environment can be expected to show up in more general topologies. Rings also exist as subgraphs of more elaborate distributed networks. Hence, algorithms designed for rings have applications in networks with richer topologies. Consequently, rings serve as a suitable testbed in which to sharpen our intuitions, develop basic tools, and gain insight into factors that influence the complexity of communication.

The assumption of distinct identifiers is the usual default within the domain of deterministic algorithms because identifiers are typically (though by no means always) required to solve problems deterministically. In contrast, the focus here is on *anonymous* networks: that is, networks where distinct node identifiers cannot be guaranteed (or where they are disregarded for the sake of generality). Randomization is employed to skirt the limitations of determinism on an anonymous ring. With the help of randomization all recursive functions can be computed on an anonymous ring of known size. Furthermore, even when a deterministic solution exists, randomization frequently provides a more efficient one.

Even within the restricted anonymous ring configuration, there are a number of additional parameters that can be seen to influence the inherent complexity of fundamental problems. Close study of specific problems serves to highlight a number of these issues. By designing distributed algorithms, it becomes apparent that small changes in the requirements of an algorithm seem to have a significant influence on its communication complexity. Lower bounds confirm that the perceived sensitivity to additional features of the model is real.

One such parameter is the type of program that is being run by the processors of the network. The programs may or may not incorporate randomization. If randomization is employed, the resulting algorithm may be required to eventually terminate and to be correct upon termination. Alternatively, the less stringent requirement, that with high probability the algorithm terminates correctly, may be imposed. For some problems,

the inherent complexity decreases with each generalization from one of these models to a more powerful one. Randomization, however, is not a panacea. There are other problems whose communication complexity is not affected by more than a constant factor even when randomized solutions that err with small probability are permitted.

Algorithms required to work for all possible topologies can be expected to have a higher complexity than ones designed for a fixed topology. It is perhaps less obvious that even after restricting the network to an asynchronous unidirectional ring of indistinguishable processors, the complexity of a given problem is not necessarily determined. Algorithms that are designed for a fixed size ring might have a lower complexity than any algorithm that must work for a larger class of rings. The degree of this sensitivity to knowledge of the ring size is problem specific.

A third factor influencing complexity is the type of termination required of an algorithm. Usually it is assumed that a processor ceases computation after arriving at a conclusion. This means that subsequent messages cannot influence this conclusion. This is called *distributive termination.* A weaker notion of termination permits processors to reach tentative conclusions which may be revoked upon receipt of further communication. Hence, the conclusions are only final when all message traffic has ceased, a situation which may not be detectable. This type of termination is called *nondistributive termination.* For some problems, the lower bounds under the assumption of nondistributive termination can be achieved by distributively terminating algorithms. The complexity of other problems is influenced by the type of termination required.

No doubt additional features of a model might also be considered. However, these three parameters — algorithm type, knowledge of ring size, and type of termination — are sufficient to illustrate the rich spectrum of issues that influence communication complexity. For each of these parameters, there are problems whose complexities are sensitive to the specific assumptions and other problems whose complexities are not.

Many fundamental problems on rings with various additional assumptions have been widely studied in the absence of randomization. However, it has become increasingly

apparent that randomization can be a powerful tool particularly when used as an aid to breaking symmetry. So it is natural to explore the effect of randomization on the complexity of algorithms for an asynchronous distributed environment. A central thesis of this dissertation is that randomization can contribute to the design of simple and efficient distributed algorithms. In defence of this claim, it will be demonstrated that on a ring:

1. Randomized algorithms solve some problems in distributed computing that cannot be solved by deterministic algorithms.

2. Some problems in distributed computing that have deterministic solutions, have randomized solutions which significantly reduce the communication complexity over the best possible deterministic solutions.

3. Randomization is frequently natural and easy to employ, resulting in algorithms that are conceptually straightforward and are easily proved correct.

In addition, randomization admits the possibility of probabilistic solutions to distributed problems. On occasion it may be advantageous to tolerate a small probability of error in a solution in exchange for increased efficiency. In other situations there may be no algorithm that solves a specific problem with certainty, however randomized algorithms exist that "almost certainly" provide a correct solution. Such an algorithm necessarily employs randomization, for any reasonable definition of "almost certainly".

There are two standard terms for describing these two uses of randomization [11]. A *Las Vegas* algorithm is required to terminate with probability one and to be correct when termination does occur. A *Monte Carlo* algorithm must have probability at least $1 - \epsilon$ of terminating correctly, where ϵ is a parameter of the algorithm.

Certain properties shared by several distributed computing algorithms point to explanations for the effectiveness of randomization. It has been pointed out that the asynchronous nature of distributed computation contributes to the difficulty of designing correct algorithms. So it may seem paradoxical that the computations with highest communication complexity often arise during executions that happen to proceed synchronously. The syn-

chronous execution preserves processor symmetry. But very few exchanged coin tosses can be expected to break symmetry. Hence, randomization quickly overcomes the problems created by local symmetry. Deterministic distributed computing solutions frequently include the exchange of processor identifiers. Typically, only locally distinct rather than globally distinct identifiers are required for part of an algorithm. A constant number of random bits at each processor can be expected to distinguish processors locally. So it might be possible to replace exchanges of long identifiers with exchanges of short sequences of random bits, thus achieving an effective algorithm with lower bit complexity. If the resulting algorithm makes no use of the processor identifiers, then as a bonus, the new algorithm can be applied to rings that cannot guarantee distinct processors.

A related observation deserves mention here. Sometimes a simple deterministic solution to a distributed computing problem performs well on average. However, for a small proportion of the possible assignments of identifiers to processors, this natural solution has a high communication complexity. Eliminating poor worst case performance while maintaining a deterministic solution could require a significant increase in the intricacy of the code. An obvious alternative is to employ randomization while keeping the structure of the original simple algorithm. The association between particular inputs and expensive computations is broken by randomization. For any particular input, the worst case scenario occurs with only very low probability. Consequently, the resulting randomized algorithm maintains a low expected complexity. Algorithms that use randomization to achieve this averaging effect for all identifiers or inputs have been called *Sherwood* algorithms [11].

Once it is determined that randomization can be helpful, it is natural to ask "How much does randomization help?". This opens the question of lower bounds for randomized asynchronous computation. Lower bounds for randomized algorithms are typically more difficult to establish than the corresponding deterministic ones, even in the sequential setting. In the distributed setting, the combined factors of asynchrony and randomization pose substantial barriers to general lower bound techniques.

On a unidirectional ring, one of the barriers due to asynchrony can be circumvented. In a general asynchronous network the scheduler has some control of the ordering of mes-

sages arriving at a processor over different links. This power is lost in a unidirectional ring because messages are received on only one link. By generalizing from randomized to nondeterministic algorithms, the barrier due to randomization can be finessed at times. (The automata theoretic notion of nondeterminism is intended throughout this dissertation. An algorithm provides a nondeterministic solution to a problem if — loosely — it never results in a wrong answer to the problem. The complexity of such an algorithm on a fixed input is the minimum complexity over all correct computations for that input.) Nondeterminism can be considered to model the best case execution of a randomized algorithm. These best case lower bounds then apply to the less powerful randomized model.

Recently, Duris and Galil [14] introduced some new techniques for proving average case lower bounds for deterministic algorithms for rings. These can be adapted to apply to the expected case of randomized algorithms. The techniques provide an alternative for proving lower bounds for randomized solutions when nondeterministic lower bounds are too weak.

Two recurrent themes serve to unify the material in subsequent chapters. One is the effect of randomization on the design of algorithms for asynchronous distributed systems. Another is the surprising influence that the secondary model characteristic, knowledge of ring size, has on the complexity of some fundamental problems. An overview of the remaining chapters follows.

If one processor in a network is distinguished as a leader, then that processor can coordinate further computation. For this reason, leader election has long been recognized as a fundamental problem in distributed computing. The bulk of research into this problem has addressed deterministic solutions to leader election. Las Vegas leader election, however, offers significant advantages over deterministic election. Chapter 2 describes one such algorithm for leader election on a ring. The solution has evolved from an earlier Las Vegas solution (see [2] and [1]). The variant in chapter 2 is conceptually simpler and has an easier proof of correctness than its predecessor, while maintaining the same expected complexity.

Chapter 3 contains confirmation of the fundamental nature of leader election. Deterministic solutions for other basic problems such as ring orientation and election on networks

related to rings have received attention [9,10,19,27]. These problems are re-examined in chapter 3 with the intent of determining the contribution of randomization. Las Vegas solutions are constructed from the leader election algorithm of chapter 2. These solutions inherit the advantages that Las Vegas leader election has over deterministic leader election. In particular, the bit complexities of the resulting algorithms are lower than those of the corresponding deterministic solutions and the requirement for distinct identifiers is eliminated.

A precise model of computation is needed in order to study lower bounds for algorithms on rings. Because either messages or bits is the resource of interest, such a model should highlight the communication performed by the algorithm while making local processing transparent. A class of models meeting this criterion is presented in chapter 4. All the models are similar, but are tuned to reflect the various kinds of algorithms — deterministic, Las Vegas, Monte Carlo and nondeterministic. (Although nondeterministic algorithms are not an option for realistic solutions, the nondeterministic model is useful because lower bounds in this model imply lower bounds for the best case execution of Las Vegas algorithms.) Additional properties such as knowledge of ring size, type of termination and existence of identifiers are all captured by the models in a natural way. As will be seen, the collection of models proves to be effective for investigating the impact of these parameters on communication complexity.

The model for nondeterministic algorithms can be exploited to prove lower bounds on the bit complexity of various problems on rings. A general lower bound occurs in chapter 5. It is shown that if f is any cyclic nonconstant function, then any nondeterministic algorithm that evaluates f on a ring of size n requires at least $\Omega(n\sqrt{\log n})$ bits. The same chapter demonstrates that the lower bound is the best possible for this generality. A cyclic nonconstant function is described that can be evaluated in $O(n\sqrt{\log n})$ expected bits. This is to be contrasted with the deterministic case where the corresponding complexities are known to be $\Omega(n \log n)$ and $O(n \log n)$ bits respectively [21]. Again, randomization is seen to reduce communication complexity.

The general lower bound of chapter 5 is not tight for algorithms that evaluate natural

functions such as AND. The complexity of algorithms for AND that are required to work on all rings within a range of sizes is determined in chapter 6. The model developed in chapter 4 is used to show that even the best case of any Las Vegas algorithm for AND does not improve upon the expected complexity of the specified Las Vegas algorithm.

The lower bound in chapter 6 does not extend to algorithms that need only work on fixed size rings. This problem is addressed in chapter 7. Again the nondeterministic model can be employed; this time it is shown that, with exact knowledge of ring size, the nondeterministic complexity of AND drops. All lower bounds for Las Vegas algorithms quoted so far use techniques that apply to the best case as well as to the expected case complexity. Modifications of techniques introduced by Duris and Galil [14] are capable of distinguishing between the complexity of best case and expected case computation. They are employed to show that the expected complexity of AND does not decrease with exact knowledge of ring size. The techniques have the added advantage of easily extending from Las Vegas to Monte Carlo algorithms. So the results are presented in this more general setting.

Chapter 8 summarizes the various tools and techniques used throughout the dissertation as well as to restate and interpret the results. As might be expected, many additional enticing problems were uncovered in the process of this research. Chapter 8 outlines some of these open problems together with the partial results of an initial investigation.

Chapter 2

A Simple Las Vegas Leader Election Algorithm

Leader election causes a unique processor, from among a specified subset of the processors, to enter a distinguished final state. This problem is one of several problems which are fundamental in that their solutions form the building blocks of many more involved distributed computations. The complexity, measured by the number of communication messages, of leader election on distributed rings with various combinations of properties has been well studied, as described below.

On an asynchronous unidirectional ring of n processors with distinct identifiers, a leader can be elected by a deterministic algorithm, which operates using pairwise comparisons of processor identifiers, using $O(n \log n)$ messages each of $O(m)$ bits [13,25], where m is the number of bits in the largest processor identifier. If the universe of identifiers is unbounded, any deterministic leader election algorithm for an asynchronous ring that does not employ knowledge of ring size, must exchange $\Omega(n \log n)$ messages (of arbitrary length) in the worst case [12,24] or average case [24], even if bidirectional communication is possible.

If processors are not endowed with distinct identifiers then, as was first observed by Angluin [8], deterministic algorithms are unable to elect leaders, even if n is known to all processors. Itai and Rodeh [17] propose the use of randomized algorithms to skirt this

limitation. They present a randomized algorithm that elects a leader in an asynchronous ring of known size n using $O(n \log n)$ expected messages of $O(\log n)$ bits each. The lower bound results of [23] show that even if processors have distinct identifiers drawn from some sufficiently large universe, the expected number of messages (of arbitrary length) communicated by a randomized leader election algorithm is $\Omega(n \log n)$ as long as the algorithm is required to work for a large range of ring sizes. With respect to bit complexity, however, the algorithms cited above for asynchronous rings are not optimal. A randomized algorithm to elect a leader on a unidirectional asynchronous ring with complexity matching the message complexity of the algorithm in [17], appears in [2]. In this algorithm, ring size need only be known to within a factor of two and the low message complexity is achieved using constant length messages. A matching lower bound of $\Omega(n \log n)$ bits is also contained in [2] proving that no significant bit complexity improvement is possible in this model. The proof of this lower bound requires some flexibility in ring size. In particular, it does not apply when the ring size is known exactly.

These results can be contrasted with the results achieved when communication is guaranteed to proceed synchronously. If interprocessor communication is synchronous and identifiers are drawn from some known countable universe, then $O(n)$ messages suffice to elect a leader in the worst case [15]. But this reduced complexity can be achieved only at the expense of time. Even if the ring size n is known to all processors, deterministic algorithms that are restricted to operate either by comparisons of processor identifiers or within a bounded number of rounds, must transmit $\Omega(n \log n)$ messages in the worst case [15]. However, $O(n)$ expected messages suffice for randomized leader election on a synchronous anonymous ring [17], provided the ring size n is known to all processors.

An alternative algorithm for randomized leader election on an anonymous ring with message and bit complexity matching that of [2] is described in this chapter. The algorithm has evolved from this earlier leader election algorithm, and is most naturally described in an analogous manner. For this reason, much of what follows parallels the development in [2].

2.1 Leader Election, Attrition and Solitude Verification

Leader election requires that a single processor be chosen from among some nonempty subset of processors called *candidates*. Initially each candidate is a *contender*. A leader election algorithm must i) eliminate all but one contender by converting some of the contenders to *noncontenders*, and ii) confirm that only one contender remains. This suggests the separation of leader election into two subtasks called attrition and solitude verification respectively (cf.[2]). More formally, a procedure solves the *attrition problem* if, when initiated by every candidate, it never makes all candidates noncontenders, and with probability 1 it takes all but exactly one of these candidates into a permanent state of noncontention. Typically an attrition procedure does not terminate but rather enters an infinite loop in which the remaining contender continues to send messages to itself. An algorithm solves the *solitude verification* problem if, when initiated by a set of processors, it terminates with probability 1 and upon termination an initiator is in state "yes" if and only if there is exactly one initiator. An algorithm for the related problem, *solitude detection*, must terminate with probability 1, and upon termination, if there is exactly one initiator then the initiator is in state "yes", and if there is more than one initiator then all initiators are in state "no". If P is a predicate on algorithms for rings, then the term *eventually* P is used to abbreviate the phrase *terminates with probability* 1 *and upon termination* P. Under this convention, an algorithm solves solitude verification if, when initiated by a set of processors, eventually an initiator is in state "yes" if and only if there is exactly one initiator. Similarly, if an algorithm solves solitude detection then, eventually, if there is exactly one initiator then the initiator is in state "yes", and if there is more than one initiator then all initiators are in state "no".

The relationship between leader election and the two subtasks, attrition and solitude verification, was pointed out and exploited by Itai and Rodeh [17]. By tightly interleaving the solutions to the two subproblems, the algorithm in [2] achieves a lower expected bit complexity than that of the algorithm in [17]. The variant of the algorithm in [2] that is described below employs a different interleaving strategy which can be implemented with

a simpler attrition procedure. As a result, the proof of correctness is simplified.

Attrition and solitude verification are interleaved to solve leader election by annotating each attrition message with one solitude verification message. Whenever a contender enters a state of noncontention, it forwards a solitude verification restart message to alert remaining contenders that they were not previously alone. When attrition has reduced the set of contenders to one, solitude verification will proceed uninterrupted, verifying that only one contender, the *leader*, remains.

The efficiency of the leader election algorithm depends not only on the efficiency of the attrition and solitude verification procedures from which it is constructed, but also on the cost of interleaving. If solitude verification is *conservative* in the sense that every message is bounded in length by some fixed constant number of bits, then annotated attrition messages are at worst a constant factor longer than unannotated messages. So the cost of premature attempts to verify solitude is dominated by the cost of attrition. Similarly, if attrition messages have constant length, then the cost of the attrition messages that are sent after attrition has reduced the number of contenders to one, is at most a constant factor more than the cost of the solitude verification algorithm. The only remaining cost attributable to interleaving involves the transmission of restart messages. As will be seen, this cost is subsumed by the cost of attrition.

The next three sections (2.2 through 2.4) demonstrate how to achieve each of the three tasks: attrition, solitude verification and interleaving on an asynchronous unidirectional ring. Processors are message-driven. First some subset of processors each initiate one message. Computation proceeds by each processor repeatedly executing the steps: (1) block until some message is received, (2) do the specified local processing, and (3) send the message (possibly null) determined by the local processing. The goal is a Las Vegas leader election algorithm, and therefore the solution is required to terminate with probability 1 and upon termination to be correct. Though efficient leader election can be achieved by interleaving conservative solitude verification with efficient attrition, section 2.3 actually describes a conservative solitude detection algorithm.

2.2 The Attrition Procedure

The attrition procedure is initiated by all candidates (the initial *contenders*) for leadership. The number of candidates is denoted by c. The procedure uses random numbers to eliminate some contenders while ensuring that it is not possible for all contenders to be eliminated. Contenders create messages which are propagated to the next contender. A noncontender never converts to a contender, but behaves entirely passively — simply forwarding any messages received.

The procedure has an implicit round structure and uses a parameter k which determines message length. In each round, each contender independently generates an integer chosen randomly from the uniform distribution on the interval $[0, 2^k - 1]$, sends the outcome to its contending successor and waits to receive the random number generated by its contending predecessor. The processor becomes a noncontender for the remainder of attrition if and only if it sent a number smaller than the one it received. Those processors remaining in contention proceed with the next round.

Correctness: Let $\pi_1, \ldots, \pi_m$, $m \geq 1$, be the contenders at the beginning of an arbitrary round, j, and s_i be the random number sent by π_i in round j. Suppose $s_k = \max\{s_1, \ldots, s_m\}$. The corresponding processor π_k must remain a contender for round $j + 1$. Therefore, not all processors can become noncontenders. On the other hand, the probability that a given contender sends a message that is smaller than the one it receives is a function of k that is clearly at least $1/4$ as long as there is more than one contender. Hence, with probability 1, the number of contenders decreases to one.

Complexity: When only one contender remains, it continually receives the same random number as it last sent. This infinite loop is broken only by the intervention of solitude verification. Therefore, the complexity of concern for the analysis of attrition is the expected number of bits expended until the number of contenders is reduced to 1. Let $\pi_1, \ldots, \pi_m$ be the contenders at the beginning of an arbitrary round j of attrition. Define the random

variables $Y_i, 1 \leq i \leq m$, for this round by:

$$Y_i = \begin{cases} 1 & \text{if } \pi_i \text{ is a contender at the beginning of round } j+1 \\ 0 & \text{if } \pi_i \text{ is a noncontender at the beginning of round } j+1 \end{cases}$$

Given that there are at least 2 contenders in round j, the k-bit random number generated by contender π_i in round j is independent of the random number produced by its nearest preceding contender. Therefore, $\mathrm{E}(Y_i|m \geq 2) = (2^k+1)/(2^{k+1})$. Let random variable X_j be the number of contenders at the beginning of round j. Then $X_1 = c$. So, if $m \geq 2$:

$$\begin{aligned} \mathrm{E}(X_{j+1}|X_j = m) &= \mathrm{E}\left(\sum_{i=1}^{m} Y_i\right) \\ &= \sum_{i=1}^{m} \mathrm{E}(Y_i) \\ &= \sum_{i=1}^{m} \frac{2^k+1}{2^{k+1}} \\ &= m \cdot \frac{2^k+1}{2^{k+1}} \end{aligned}$$

And $\mathrm{E}(X_{j+1}|X_j = 1) = 1$. Therefore:

$$\begin{aligned} \mathrm{E}(X_{j+1}) &= \sum_{m \geq 1} \mathrm{E}(X_{j+1}|X_j = m) \cdot \Pr(X_j = m) \\ &= \sum_{m \geq 2} \mathrm{E}(X_{j+1}|X_j = m) \cdot \Pr(X_j = m) + \Pr(X_j = 1) \\ &= \sum_{m \geq 2} \left(m \cdot \frac{2^k+1}{2^{k+1}} \cdot \Pr(X_j = m)\right) + \Pr(X_j = 1) \\ &= \frac{2^k+1}{2^{k+1}} \sum_{m \geq 1} m \cdot \Pr(X_j = m) + \left(1 - \frac{2^k+1}{2^{k+1}}\right) \Pr(X_j = 1) \\ &= \frac{2^k+1}{2^{k+1}} \mathrm{E}(X_j) + \left(1 - \frac{2^k+1}{2^{k+1}}\right) \Pr(X_j = 1) \end{aligned}$$

Thus, after $r = \log_{\frac{2^k+1}{2^{k+1}}} c = (1 - \log(1+2^{-k}))^{-1} \log c$ rounds of attrition, $\mathrm{E}(X_r)$, when c candidates initiate attrition, is a small constant (at most 5)[1]. The expected number of rounds required to reduce from 5 contenders to one is a constant and each round of attrition requires exactly nk bits. Therefore:

[1] Logarithms denoted by "log", without explicit bases, are assumed to be base 2.

Lemma 2.1 *The expected number of bits communicated by the attrition procedure when there are c candidates, up to the point where there is only one remaining contender, is at most* $(1 - \log(1 + 2^{-k}))^{-1} kn \log c + O(n)$, *where k is the number of bits in each message.*

For some applications, it will be convenient to use the attrition procedure with parameter k set to 1. To highlight the simplicity of this version, called *simple attrition*, the single bit random numbers are referred to as random coin tosses which generate elements of $\{h,t\}$.

Corollary 2.2 *The expected number of bits communicated by the simple attrition procedure with c candidates up to the point where there is only one remaining contender is at most* $n \log_{\frac{4}{3}} c + O(n) < 2.41 n \log c + O(n)$.

The expected *message* complexity of the attrition procedure is at most $(1 - \log(1 + 2^{-k}))^{-1} n \log c + O(n)$. Notice that this can be brought arbitrarily close to $n \log c + O(n)$ by increasing k, the length of the attrition messages. Since $(1 - \log(1 + 2^{-k}))^{-1} < 1 + 2\log(1 + 2^{-k}) = 1 + (2/\ln 2)\ln(1 + 2^{-k}) < 1 + (2/\ln 2)2^{-k}$, the complexity of the attrition procedure described here is less than $(1 + 2^{-k+2}) n \log c + O(n)$ expected messages[2].

2.3 The Solitude Detection Algorithm

In the absence of any information about the ring, even solitude verification is impossible (cf. [8,2]). Therefore, solitude verification algorithms must use specific ring information to verify that there is a sole contender. On an anonymous ring of known size, solitude can be verified by confirming that the gap between a contender and its nearest preceding contender is equal to the ring size. Though it is sufficient to combine solitude verification and attrition to achieve leader election, the following describes a solitude detection algorithm.

Suppose that each processor knows the size n of the ring. A nonconservative algorithm for determining solitude has each contender initiate a counter which is incremented and

[2] Logarithms denoted by "ln" stand for the natural logarithm, base e.

forwarded by each noncontender until it reaches an initiator, π_x. By comparing the received counter with n, π_x knows whether or not it is alone. This algorithm can be transformed into a conservative solitude detection algorithm without any increase in bit communication complexity.

Each processor π_x, whether contending or noncontending, maintains a counter c_x, initialized to 0. Let $d_x > 0$ denote the distance from π_x to its nearest preceding contender. The algorithm maintains the invariant:

if π_x has received j bits then $c_x = d_x \bmod 2^j$.

Then if π_x reaches a state where $c_x = n$, there must be $n-1$ noncontenders preceding π_x, so π_x can conclude that it is the sole contender. It remains to describe a strategy for maintaining the invariant.

Contenders first send 0. Thereafter, all processors alternately receive and send bits. If π_x is a noncontender, then π_x is required to send the j^{th} low order bit of d_x as its j^{th} message. Contenders continue to send 0. Suppose a processor, π_y, has the lowest order $j-1$ bits of d_y in c_y. A simple inductive argument shows that when π_y receives its j^{th} message (by assumption the j^{th} bit of $d_y - 1$), it can compute the first j bits of d_y and thus can update the value of c_y to satisfy $c_y = d_y \bmod 2^j$.

In the previous algorithm it was assumed that n is known exactly. Suppose instead that each processor knows an integer N, such that $N \leq n \leq 2N-1$. Then there can be at most one gap of length N or more between neighbouring contenders. Thus, any gap of less than N confirms nonsolitude. Any processor detecting a gap of $m \geq N$ can determine solitude by initiating a single round to check if the next gap is also m. (For the purposes of leader election, it is sufficient for any contender that detects a gap of N or more to declare itself the leader, since it has confidence that no other processor can do the same.) The modified algorithm is a correct solitude detection algorithm when ring size is known to within a factor of less than two.

Lemma 2.3 *If each processor knows a value N such that the ring size n satisfies $N \leq n \leq 2N-1$, then conservative and deterministic solitude detection can be achieved using at most $O(n \log n)$ bits.*

2.4 Interleaving Attrition and Solitude Detection

2.4.1 A simple leader election algorithm

A leader can be elected on a ring of size $n \in [N, 2N-1]$ with just a coarse interleaving of the attrition procedure and a trivial solitude detection algorithm. First, attrition is run until, with overwhelming probability, there is one remaining contender. This is followed by a single round of solitude detection where each remaining contender sends a counter to measure the gap between itself and its nearest preceding contender. In the rare event that nonsolitude is confirmed, these two steps are repeated by the remaining contenders until solitude is verified. The algorithm is stated for contenders; noncontenders participate by forwarding attrition messages and incrementing and forwarding gap-counting messages.

```
Algorithm SIMPLE-LE:
1. WHILE contender AND NOT leader DO
2.     r rounds of simple attrition;
3.     IF contender THEN
4.         send(< 0 >);
5.         receive(<counter>);
6.         IF counter ≥ N THEN leader ←true.
```

SIMPLE-LE will eventually elect a leader for any value of r greater than or equal to 1. Suppose that C is an upper bound on the number of candidates for leadership. Each execution of lines 3 through 6 uses $n \log n$ bits. If, in line 2, r is set to $a \log C$ for some $a \geq 1$, then the expected number of times through the WHILE loop will be constant. Thus, the detection portion will contribute an expected complexity of $O(n \log n)$ bits. For this value of r, the attrition portion contributes $O(n \log C)$ bits to the expected complexity of this algorithm. The upper bound on the number of candidates is used to ensure that premature checking for solitude is unlikely. Algorithm *AND2* on page 79 in chapter 6 uses

a similar strategy to compute AND on a ring. The analysis of *SIMPLE-LE* is a simplified version of the analysis of *AND2* and is therefore omitted.

If the value of r used in line 2 is much larger than a constant times the logarithm of the actual number of candidates, then the attrition step of this simple algorithm is inefficient. If it is too small, then inefficiency (in terms of bit complexity) results from premature solitude checking. For *SIMPLE-LE* to be efficient, some sufficiently accurate estimate on the number of candidates is required. Also, this algorithm cannot be adapted to one that provides an efficient solution to leader election when there are distinct identifiers but no knowledge of ring size nor of the number of candidates. These shortcomings are overcome by a tighter interleaving of attrition and solitude detection messages. The remainder of this section describes this more elaborate algorithm.

2.4.2 Informal description of interleaving

Section 2.3 describes a conservative solitude detection algorithm for a static situation. It assumes that there are fixed contenders and noncontenders on the ring, and describes an algorithm such that each processor, π_x, can determine the bits of its distance, d_x, to its nearest contending predecessor. However, when interleaved with attrition, the situation is not static since contenders turn to noncontenders as the attrition progresses. *SIMPLE-LE* avoids the resulting complication because it collects complete gap information each time solitude is checked. In contrast, if each round of attrition is interleaved with one round of conservative solitude detection, then gaps between contenders are likely to combine into longer gaps before complete gap information is collected. Therefore, it becomes necessary to signal when a processor's accumulated gap information is no longer relevant. This is done with a restart flag set by noncontenders that were contenders in the previous round.

In each round, a message with 3 fields (k attrition bits, a solitude detection bit, and a restart flag initialized to false) is sent by each contender and propagated to the next contender. If a message arrives at a noncontender, π_p, that was a contender in the previous round (a *new noncontender*), then all processors following π_p, up to and including the next

contender, have gap information that is no longer correct. Noncontender π_p signals this situation to these successors by setting the restart flag in the message. Processors that receive a message with the restart flag set, reinitialize their solitude detection variables. Note that the first new noncontender following a contender retains correct gap information, since the gap preceding this new noncontender remains unchanged. In previous rounds, this new noncontender accumulated some bits (at least one) of this unchanged gap. Therefore, it can send the first bit of that gap as the required bit in the solitude detection field of its message. The bit position of the outgoing solitude detection bit will lag behind the bit position of the incoming solitude detection bit for the new noncontender until it receives a message with the restart flag set.

2.4.3 The leader election algorithm

The leader election algorithm presented in this subsection is designed for asynchronous anonymous rings of size $n \in [N, 2N-1]$. Recall that processor π_i maintains the following two local variables which are described in section 2.3. The gap from processor π_i to its contending processor is denoted d_i.

j_i: count of the number of messages received containing correct gap information.

c_i: gap counter containing the value $d_i \bmod 2^{j_i}$.

For noncontenders, the outgoing bit position for solitude detection may be less than the incoming bit position. Therefore, an additional variable, o_i, representing the position of the outgoing solitude detection bit, is introduced. For clarity, the following functions and procedures which carry out conservative solitude detection as described in section 2.3 are assumed as subroutines.

leader: a boolean function that returns true if and only if the local variable c_i has a value in $[N, 2N-1]$.

gapupdate(x): Increments the position counters j_i and o_i. Then uses bit x to update the gap information in the counter c_i in order to maintain the invariant $c_i = d_i \bmod 2^{j_i}$.

initializesv: sets gap variables to their initial values in preparation for processing the first bit of gap information. ($j_i \leftarrow 0$, $o_i \leftarrow 0$ and $c_i \leftarrow 0$.)

nextsvbit(b): produces the b^{th} bit of the number d_i given that $c_i = d_i \bmod 2^{j_i}$ and $j_i \geq b$.

random(x): puts a random number of fixed length k into variable x.

```
Algorithm LE:
   initializesv; contender ←true;
   WHILE contender AND NOT leader DO
      send(<random(mynumber), 0, false>);
      receive(<prednumber, svbit, restart>);
      IF restart THEN initializesv;
      gapupdate(svbit);
      IF mynumber < prednumber THEN
         o ←0;
         contender ←false;
   WHILE NOT contender DO
      receive(<prednumber, svbit, restart>);
      IF restart THEN initializesv;
      gapupdate(svbit);
      IF o = 1 THEN restart ←true;
      send(<prednumber, nextsvbit(o), restart>).
```

Correctness: The correctness of attrition and solitude detection have been established in lemmas 2.1 and 2.3. With probability 1, attrition reduces the set of contenders to one, and solitude detection confirms that one contender is left. It only remains to prove that interleaving, using restart flags, correctly maintains gap information. Define round number r of processor π_i to be the interval in the execution of π_i after it has received r messages and before it has received $r+1$ messages. Let variable v_i^r denote the value at the end of round r of processor π_i's local copy of variable v. Let $\gamma_1^r, \ldots, \gamma_{m_r}^r$ be the processors that are contenders at the beginning of round r. Let d_i^r be the distance from π_i to the nearest predecessor in $\{\gamma_1^r, \ldots, \gamma_{m_r}^r\}$.

Claim 2.4 *For every round number, r, and for every $1 \leq i \leq n$, $c_i^r = d_i^r \bmod 2^{j_i^r}$.*

Proof: Clearly the claim holds at the end of round 1, given the correctness of solitude detection. Consider a message from contender γ_{l-1}^r to contender γ_l^r in round r over gap d_l^r

and assume that all accumulated gap information is accurate at the end of round $r-1$. If no processor between γ_{l-1}^r and γ_l^r is a new noncontender, then the gap d_i^r is unchanged from d_i^{r-1} for any processor π_i between γ_{l-1}^r and γ_l^r. The correctness of solitude detection ensures that π_i will accumulate one more bit of information about d_i^r as required. Suppose that some processors $\rho_1, \ldots, \rho_p$ between γ_{l-1}^r and γ_l^r are new noncontenders. They each have $o = 0$ and *contender*=`false` at the beginning of round r. Since **gapupdate** increments o, each of $\rho_1, \ldots, \rho_p$ sets the restart flag. The restart flag is first set by ρ_1, so all processors from γ_{l-1}^r to ρ_1 retain their gap information and acquire one more significant bit. Hence, the claim holds up to and including processor ρ_1. Processor ρ_1 has at least the first two bits of its preceding gap in its local copy of variable c. Therefore, it sends the correct first bit to its successor. All remaining processors π_x up to and including γ_l^r receive a message with the restart flag set and a correct first bit of the new gap, d_x^r. Thus, the claim holds for these processors as well. ■

Complexity: By the complexity of attrition, *LE* expends an expected $O(nk \log c)$ bits up until one contender remains. At this point, each message sent by the sole remaining contender drives one bit of gap information back to it. After $\lceil \log N \rceil + 1$ more rounds, its solitude will be confirmed. Therefore, by choosing k a constant:

Theorem 2.5 *Algorithm LE elects a leader on a ring of n processors where* $n \in [N, 2N-1]$ *using* $O(n \log n)$ *expected bits.*

LE elects a leader under the weakest possible condition on an anonymous ring since solitude cannot even be verified if ring size is not constrained as theorem 2.5 requires.

2.4.4 Election on rings with identifiers

If processors have distinct identifiers, then solitude can be verified using this information rather than knowledge of the ring size. Algorithm *LE* can easily be adapted to this situation. The result is a leader election algorithm for rings with distinct identifiers and unknown ring size.

The algorithm for conservative solitude detection on ring with distinct identifiers is the natural one. Suppose that each processor, π_x, has an identifier, I_x, that is terminated by an *end-of-identifier* marker. Processor π_x uses an internal string variable, J_x, which is initialized to the empty string. Each contender alternately sends the j^{th} bit of its own identifier and receives the j^{th} bit of its nearest preceding contender, which it appends to J_x. Thus, π_x builds up in J_x the identifier of its nearest preceding contender. If I_x contains m bits, then after receipt of at most m bits, π_x can declare, by comparing J_x and I_x, whether or not it is alone. Because no identifier is a prefix of any other identifier, π_x can never falsely claim solitude.

Lemma 2.6 *If processors have distinct m bit identifiers, then conservative and deterministic solitude detection can be achieved with distributive termination using at most $O(mn)$ bits.*

Interleaving of this solitude detection with attrition to achieve leader election is similar to algorithm *LE*. Rather than sending 0 for the solitude detection bit, each contender sends successive bits of its identifier in each round until all its identifier bits have been sent. In subsequent rounds, each contender uses an end-of-identifier marker in the solitude detection field until it either becomes a noncontender or is elected leader.

Theorem 2.7 *Algorithm LE can be adapted to elect a leader on a ring of size n where processors have distinct identifiers of length at most m bits using $O(nm)$ expected bits.*

2.5 Tuning the Leader Election Parameters

The worst case scenario, that all n processors are candidates for leadership, is assumed in this section. If the attrition used in *LE* is simple attrition, then the number of messages used for attrition alone is expected to be more than $n \log_{4/3} n > 2.4n \log n$. This exceeds the complexity of the leader election algorithm in [13] for rings with distinct identifiers. The algorithm in [13] has message complexity less than $1.356n \log n + O(n)$ and remains the deterministic leader election algorithm with the best known message complexity. But by

sending longer attrition messages, the base of the logarithm in the complexity expression for attrition can be brought within ϵ of 2. Similarly, the $n \log n$ messages sent by solitude detection after there remains only one contender, can be reduced by a factor of $l \leq \log n$ by sending l detection bits with each attrition message. By tuning these parameters, the expected message complexity of leader election can be reduced to within a factor of $(1+\epsilon)$ of $n \log n$ for $\epsilon > 0$. For example, choosing parameter $k = 4$, and thus choosing independent random integers from the uniform distribution on the interval $[0, 15]$, the expected message complexity of attrition drops to less than $1.096n \log n$. By choosing $l = 4$, the solitude detection adds only $(n \log n)/4$ messages for a total of less than $1.346n \log n$ expected messages. These settings result in a Las Vegas leader election algorithm that has lower expected message complexity than the lowest complexity known for a deterministic algorithm, while retaining both constant length messages (only 9 bits) and simplicity. Clearly, the same packaging idea can be employed whether solitude is detected using identifiers or ring size information.

Reducing the expected message complexity is not a beneficial strategy for minimizing the expected bit complexity because this reduction is done at the expense of longer messages. The optimal parameter setting for minimizing bit complexity of the leader election algorithm occurs when messages are short. If k bits are used in an attrition field, and l bits are used in the solitude detection field, then the annotated messages of LE are $k + l + 1$ bits each. First $\log_{k'} n$ rounds of attrition are expected until one contender remains, where $k' = 2^{k+1}/(2^k + 1)$. This is followed by $\left\lceil \frac{\log n}{l} \right\rceil$ rounds to confirm solitude. Therefore, the expected total number of bits is $n(k+l+1)(\log_{k'} n + \left\lceil \frac{\log n}{l} \right\rceil)$. This is minimized at $k = 2$ and $l = 2$ resulting in an expected complexity for LE of fewer than $8.88n \log n$ bits and fewer than $1.98n \log n$ messages. Of course the bit complexity can be reduced slightly further by running pure attrition for several rounds before interleaving with solitude detection, since it is known that at least $\log n$ rounds will be required anyway.

Chapter 3

Applications of Las Vegas Leader Election

As demonstrated in [17] and [2] and again in the previous chapter, leader election on an anonymous ring is made possible by randomization, although deterministic election in the same model is impossible. When a leader is determined, the initial symmetry of an anonymous ring is broken. The elected leader is able to coordinate further computation. As a consequence, some problems that have no deterministic algorithmic solutions, can be solved easily by employing randomized leader election. For some other problems, the efficient randomized leader election algorithm implies an improvement in the communication complexity over that achievable by deterministic algorithms. This chapter illustrates these phenomena with various examples.

3.1 Function Evaluation

In a distributed setting, function evaluation refers to the problem of computing the value of a fixed function of n variables where initially each processor in the network has as input, the value of one of the variables. Evaluation of common functions on rings is examined in some detail in chapters 6 and 7. The problem of determining the minimum possible

complexity of evaluating any nontrivial function on a ring is also addressed separately (see chapter 5). This section is only intended to point out that all functions on rings can by inexpensively evaluated by preprocessing with leader election.

Functions are usually thought to have a fixed number of arguments, and hence translate in the distributed setting to rings of fixed size. However, some common functions such as SUM and AND are easily generalized to functions over a domain of strings and hence can be interpreted as having an arbitrary number of inputs. Evaluating any meaningful function, even in this general setting, can be reduced to leader election in $O(n)$ messages. Once a leader is elected, the leader simply circulates a message which collects all the necessary information to compute the given function. When the message returns, the leader computes the function value locally and announces the result. Thus, there exists an algorithm for evaluating any function in any situation where a leader can be elected.

For example, SUM can be evaluated by a randomized algorithm on any ring that has size known to within a factor of two. $O(n \log n)$ expected messages and bits are used for election and $O(n)$ messages to compute the sum yielding an expected complexity of $O(n \log n)$ messages and $O(n \log n + n \log S)$ bits, where S is the sum. In contrast, SUM cannot be evaluated deterministically on any anonymous ring unless the ring size is known exactly. If size can vary by even one processor (say, size is either n or $n + 1$) then a deterministic algorithm running on a ring with inputs all equal to one, could not distinguish between a sum of n and $n + 1$ [10].

When the function being evaluated is the characteristic function of a regular set, this reduction to leader election is particularly efficient. Even without knowledge of ring size, for any regular set L, the function that recognizes L can be computed with $O(n)$ bits of communication on a ring with a leader, as is shown in [20].

As an illustration, consider AND on rings with size bounded within a factor of two. On input all ones, AND has complexity $\Omega(n^2)$ messages in the deterministic model even when ring size is known exactly [10]. Again, by employing leader election in the way described above, this can be reduced to $O(n \log n)$ expected bits when randomization is allowed, even

if ring size is known only to within a factor of two.

3.2 Ring Orientation

Attiya, Snir and Warmuth in [10] first introduced the problem of determining a consistent orientation on an anonymous bidirectional ring when processors have only local labels on their incident edges. Let $\pi_1, \ldots, \pi_n$ be a ring of identical processors, such that each processor π_i has two communication channels, a_i and b_i, each connected to one of its neighbours. The *ring orientation* problem is defined as follows. Each processor π_i must distinguish one channel in $\{a_i, b_i\}$ such that all the processors, together with the collection of these distinguished channels, form a unidirectional ring.

Attiya *et al* show that $\Omega(n^2)$ messages are required in the worst case for any deterministic ring orientation algorithm even when ring size is known exactly. They further show that there is no deterministic ring orientation algorithm for rings of known and even size. A deterministic orientation algorithm for rings of known and odd size is provided by Syrotiuk and Pachl in [29] with average complexity $O(n^{3/2})$ messages, assuming that all initial configurations of local orientations are equally likely. With the help of randomization both the complexity barrier and the impossibility barrier disappear. By using randomized leader election for a unidirectional ring, any anonymous ring of size n, where n is known to within a factor of two, can be oriented in $O(n \log n)$ expected bits.

Theorem 3.1 *Ring orientation reduces to leader election in* $O(n)$ *expected bits.*

Proof: Let LE be any algorithm for a unidirectional ring that chooses a unique leader from any nonempty set of candidates. Consider the following algorithm, which employs LE as a subroutine. In addition to the messages used by LE, the algorithm uses two message types — *leader* messages of the form <*l*, *finished*> (*l* indicates that this is a *leader* message, and *finished* is a boolean flag), and *orientation* messages. Each processor maintains a local *two-leaders* flag initialized to false.

Algorithm *ORIENTATION*:

1. Each processor, π, is initially a candidate and initiates LE by sending its first message of LE on its a link.

2. Upon receipt of an LE message on link b, π proceeds exactly as in algorithm LE and sends its response (if there is one) on its a link.

3. Upon receipt of an LE message on link a, π executes the leader election code for a noncontender and forwards its response on its b link.

4. When a processor is elected, it sends a *leader* message on its a link with the *finished* flag set to true.

5. A leader message is forwarded around the ring until it is received on link b by a processor that sent a leader message. (This recipient is necessarily the originator of the message.) However, any processor receiving a leader message on its a link, sets the *finished* flag to false and sets its local *two-leaders* flag to true before forwarding the leader message on its b link.

6. When a leader message returns to its originator (necessarily on a b link), this processor examines the *finished* flag. If *finished* = true then this leader propagates a final *orientation* message on its a link. If *finished* = false, then this leader delays until its local *two-leaders* flag is true.

7. When *two-leaders* is true, both leaders exchange independent random coin tosses until they send and receive opposite tosses. The leader sending heads and receiving tails propagates an *orientation* message on its a link.

8. The orientation message is forwarded until it returns to the originator of the message. Each recipient sets its *incoming* link to be the one on which the orientation message is received.

Correctness: Let A (respectively, B) be the subset of processors initially consistent with a clockwise (respectively, counterclockwise) orientation. Steps 1 through 3 perform leader

election simultaneously on sets A and B. Since the messages of each election propagate in opposing directions, any interleaving of these two elections cannot disturb their progress. Because at most one of sets A and B is empty, eventually either one or two leaders are elected. Step 6 ensures that an elected leader learns which is the situation for the current computation. If either A or B is empty, step 6 ensures that the sole leader's orientation is adopted by all processors. If neither A nor B is empty, then the delay in step 6 ensures that set A and set B have both selected a leader before the algorithm proceeds. It is then straightforward to check that the number of leaders is reduced from two to one (step 7) and that the ring is oriented consistently with this remaining leader (step 8).

Complexity: One orientation message and at most two leader messages, all of constant length, are propagated once each around the ring accounting for $O(n)$ bits. It is expected that a constant number of exchanges of coin tosses are required to select one leader from two, accounting for an additional $O(n)$ expected bits. All other messages are messages of *LE*. ■

Corollary 3.2 *An unoriented ring of size* $n \in [N, 2N-1]$ *can be oriented in* $O(n \log n)$ *expected bits using a Las Vegas algorithm.*

Although the algorithm is described for the situation when all processors start simultaneously, it is easily converted to one tolerating arbitrary wake-up. Processors that have not initiated the algorithm when a first message arrives simply adopt the role of a noncontender for leadership.

The orientation algorithm demonstrates that lack of orientation in a bidirectional ring does not significantly complicate the problem of leader election — a leader can still be elected in $O(n \log n)$ expected bits.

3.3 Leader Election in Oriented Complete Graphs

The problem of deterministically electing a leader in a complete network of distinct processors has complexity $\Theta(n \log n)$ messages. This result appears in [18] and an alternative

upper bound in [7]. With an additional condition however, the lower bound can be violated. Loui, Matsushita and West [19] and Sack, Santoro and Urrutia [27] studied a version of election on a complete network in which edge labels reflect distance information. Consider a complete network, in which each processor labels its incident edges with the numbers 1 through $n-1$. Let $f(\pi,k)$ denote the processor connected to π via π's link numbered k. The labelling is *consistent* if and only if $f(f(\pi,k),l) = f(\pi,(k+l) \bmod n)$ for every processor π and for every $0 \leq k,l \leq n-1$. The model assumed in [19,27] is an asynchronous, bidirectional complete network of processors with unique identifiers and consistently labelled incident links. The "sense of direction" constraint in [19] is equivalent to the consistent labelling property. Given this model, [19] presents a leader election algorithm with communication complexity only $O(n)$ messages. The messages used in their solution contain both identifiers and link numbers and thus may have order $\log n$ bits each.

Call a network *oriented* if all edges that are present in the network satisfy the consistent labelling constraint. Using randomization, leader election can be solved in $O(n)$ expected bits on an oriented complete asynchronous network, even if processors lack unique identifiers. The randomized algorithm is similar to the leader election algorithm for unidirectional rings of chapter 2, but after each round of attrition, the ring is updated so that subsequent messages need not pass through passive processors. Rather, each contender communicates directly with its nearest contending neighbours. Eventually only one contender remains, and the ring is updated to just a self-loop at that survivor. Thus, this processor's solitude is confirmed automatically. No explicit solitude verification is required. Conceptually, each phase of the algorithm has two parts:

1. a round of simple attrition on the current ring — contenders send and receive a single coin flip, and

2. ring revision to bypass the processors just eliminated in the attrition round.

A more detailed description follows. The algorithm is executed by each processor. The instruction **send**($< m >: l$) sends message $< m >$ on the link with local label l. The instruction **receive**($< m >$) is a blocking receive; the processor waits until a message

arrives on some channel. There are two different types of messages — attrition messages containing a random coin toss and a link number, and gap messages containing a link number or "0". Contenders process these messages alternately: if a message of one type arrives while a processor is waiting for the other, it saves this message and continues to wait. It is a consequence of the algorithm that a processor never has more than one message in a "holding" buffer. The function **random**(x) is assumed to return an unbiased random coin toss and to store the result in variable x.

```
Algorithm A:
   sendlink ←1;
   REPEAT:
1.    send(<random(sflip), sendlink> :  sendlink);
2.    receive(<rflip, receivelink>);
3.    IF NOT (sflip = t AND rflip = h) THEN
4.       send(<0> :  N−receivelink);
5.       receive(<newgap>);
6.       sendlink ←(sendlink + newgap) mod n
   UNTIL (sflip = t AND rflip = h) OR sendlink = 0
7. IF sendlink = 0 THEN
8.    announce ''elected''
   ELSE
9.    receive(<0>);
10.   send(<sendlink> :  N−receivelink).
```

Correctness: Let r_i be the total number of times that processor π_i enters the repeat loop of algorithm A. Define *round* j for π_i , $j \leq r_i$ to be the j^{th} pass of π_i through the loop. Say that π_i is *active for round* j whenever $j \leq r_i$ and that π_i is *active for* r_i *rounds.* If π_i satisfies the condition "(*sflip* = t AND *rflip* = h)" at the end of round r_i, then say that π_i *went passive* in round r_i. Let $\pi_1^j, \ldots, \pi_{n_j}^j$ denote the substring of $\pi_1, \ldots, \pi_n$ consisting of those processors that are active in round j. Then clearly $\pi_1^1, \ldots, \pi_{n_1}^1 = \pi_1, \ldots, \pi_n$ and $\pi_1^{j+1}, \ldots, \pi_{n_{j+1}}^{j+1}$ is a not necessarily contiguous substring of $\pi_1^j, \ldots, \pi_{n_j}^j$.

The following claim means that processors interpret the implicit round structure consistently. It can be established by induction on the round number and by tracing the effect of the communication on each active processor.

Claim 3.3 *The coin flip sent by* π_i^j *in its* j^{th} *round is received by* π_{i+1}^j *in its* j^{th} *round.*

The claim implies that rounds can be considered in isolation. Let x_i^j denote the value of variable x at the beginning of the j^{th} round of processor π_i. The first two lines following REPEAT constitute the simple attrition procedure of chapter 2. Therefore, given that $sendlink_i^j$ is the number of the link connecting π_i^j to π_{i+1}^j, eventually there will remain one active processor. In round 1, all active processors communicate along the links of a ring with $sendlink_i^1 = 1$ for all i. Suppose that π_i^j goes passive in round j. Simple attrition guarantees that two adjacent active processors cannot become passive in the same round. Therefore, π_{i-1}^j and π_{i+1}^j must be active processors in round $j+1$. Lines 9 and 6 ensure that $sendlink_{i-1}^j$ is updated to connect π_{i-1}^j to π_{i+1}^j. Alternatively, if π_i^j remains active for round $j+1$, then it sends a 0 (line 4) to π_{i-1}^j so that again π_{i-1}^j connects to its successor via link number $sendlink_{i-1}^j$. Finally, when one active processor remains, its *sendlink* value must be 0 ensuring correct termination.

Complexity: On first inspection of the algorithm, it may appear that messages are generally $O(\log n)$ bits long since link numbers are sent as part of the message. However, large link numbers are only sent toward the end of the algorithm, when fewer processors are active. This is enough to save a factor of $\log n$ bits from the naive complexity analysis.

Specifically, in round 1, $sendlink_i^1 = 1$ for all i and it can at most double in each round. Therefore:

Claim 3.4 *At the beginning of round j $sendlink_i^j \leq 2^{j-1}$ for every active processor π_i^j.*

Let X_j be the number of active processors in round j. From the analysis of the attrition procedure in chapter 2, $E(X_j) = \max\{(3/4)^{j-1}n, 1\}$. Each active processor in round j sends at most $1 + 2\log(2^j) < 2(j+1)$ bits. Therefore, the expected bit complexity is bounded above by:

$$E(\sum_j X_j \cdot 2(j+1)) = \sum_j (E(X_j) \cdot 2(j+1)) = \sum_j (3/4)^{j-1} n \cdot 2(j+1)) = O(n).$$

The preceding discussion is summarized by:

Theorem 3.5 *A leader can be elected in an anonymous, asynchronous, oriented complete network with n processors in $O(n)$ expected bits using a Las Vegas algorithm.*

Algorithm A does not require that a processor know which link carried an arriving message. The consistency of the labelling allows processors to compute this number from the link number used by the sender and included in the sender's message. If the model provided this information to the processors, it would be unnecessary to send link numbers with the coin tosses at line 1. The bit complexity would then drop by a constant factor but remain $O(n)$.

In a possible modification to the model, processors execute a *selective* blocking receive — that is, **receive**(k) causes a processor to wait for a message on link k. Any message arriving on link $l \neq k$ is queued. Such a message is processed only when the processor executes **receive**(l). Algorithm A can be adapted to apply to the selective blocking model. After each round of attrition, each processor must be informed of the link to its predecessor as well as to its successor so that it can selectively block on the correct link.

Claim 3.3, used in the proof of correctness of algorithm A, is perhaps somewhat subtle. One might be tempted to write the following "cleaner" algorithm:

```
Algorithm B:
   sendlink ←1;
   REPEAT:
      send(<random(sflip),sendlink> ; sendlink);
      receive(<rflip,receivelink>);
      IF (sflip = t AND rflip = h) THEN
         send(<sendlink> ; N−receivelink).
      ELSE
         send(<0> ; N−receivelink);
         receive(<newgap>);
         sendlink ←(sendlink + newgap) modn
   UNTIL (sflip = t AND rflip = h) OR sendlink = 0
   IF sendlink = 0 THEN announce ''elected''.
```

Algorithms A and B differ in the behaviour of a processor that moves from active to passive. Suppose π_a, π_b and π_c are three consecutive active processors and that π_b goes passive. (π_b sent t to π_c and received h from π_a.) In algorithm B, as soon as π_b learns that it is no longer a contender, it sends to π_a its distance to π_c. But π_b has no evidence that π_c received the t. When π_a receives the distance to π_c, it can update its own *sendlink* and proceed to the

next round. A malicious scheduler could delay the t coin flip travelling from π_b to π_c and deliver the coin flip of the following round from π_a to π_c first, since they arrive on different links. In such a scenario, the rounds of communication are not preserved and deadlock can result. Algorithm A avoids this pitfall by causing π_b to withhold the message containing link update information from π_a until it has confirmation that π_c has received its coin flip.

Algorithm B is intended to illustrate difficulties that might arise due to asynchrony. They were overcome by replacing the algorithm with algorithm A which enforces an ordering on the message delivery. What results is a simulation of a synchronous situation in an asynchronous model.

The analysis of the algorithm for leader election in an oriented complete network of processors with unique identifiers in [19], contains another example of how asynchrony can misguide our intuitions. The authors claim that no generality is lost by assuming that the algorithm proceeds in phases. Their subsequent phase by phase analysis yields a complexity of at most $3.62n$ messages. However, some generality is indeed lost with this assumption. By scheduling messages so that phase boundaries are not respected, a scenario requiring $4n$ messages is easily constructed.

Their algorithm follows. Again, **send**($< m >: l$) sends message $< m >$ on the link with local label l. Every processor has a unique identifier *id*.

```
Algorithm LMW:
   d ←n − 1;
   REPEAT
      send(<n-d, id> :  n-d);
      receive(<d, newid>)
   UNTIL newid ≥ id;
   IF newid = id THEN
      Announce ''elected''
   ELSE
      receive(<e, newid>);
      send(<n-d+e, newid> :  n-d).
```

Arrange a consistent labelling so that by following link #1 at each processor, the identifiers occur in increasing order, say 1 through n, until completing the cycle. Imagine

that the scheduler first delivers only the initial messages from processor n to 1 and from 1 to 2, delaying all others. It then delivers processor 2's identifier to processor 1 which forwards it to processor n. Processor 1 has been eliminated. After four messages have been used, the network has returned to a configuration comparable to the initial one. There are now $n-1$ active processors (processors 2 through n) and the same scheduling can be repeated with 2 replacing 1. Continuing in this way, processor n is elected after $4n$ messages.

Admittedly this is a minor error. The algorithm is correct and the complexity claim is within a small constant factor of the correct worst case complexity. However, the algorithm and its erroneous analysis serve to illustrate a recurrent difficulty with deterministic algorithms for asynchronous distributed systems. Consider a natural deterministic algorithm D which proceeds as does algorithm A with identifiers replacing coin tosses. For most identifier sequences, the computation would require only $O(n)$ messages. However, without information about the distribution of the sequences of identifiers, this does not translate into an $O(n)$ expected complexity. In general it is unreasonable to make any assumptions about the identifier sequences because they are beyond the control of the algorithm. A more elaborate algorithm is proposed in place of this natural one to circumvent the expensive worst case identifier sequences. Frequently the replacement is a complicated algorithm which is more difficult to prove correct or to analyse. (In this example, the elaboration of the natural algorithm resulted in the loss of the implicit phase structure.)

In contrast, the randomized algorithm maintains the original simplicity. It simply replaces identifiers with random coin tosses. In general, assumptions concerning the distribution of identifiers are not warranted; it is perfectly reasonable, however, to make the assumption of uniformity over sequences of random coin tosses since this assumption is supported by probability theory. A uniform distribution is precisely the extra condition required to guarantee the expected efficiency of the algorithm. Because the computation is constrained to execute in implicit rounds, a malicious scheduler is rendered impotent. An expected complexity of $O(n)$ bits results from taking the average over uniform random sequences of coin tosses and the worst case over everything else, as required for a legitimate

analysis of random computation. The fact that this is achieved with constant, rather than order $\log n$, length messages is an additional bonus of randomization.

3.4 Leader Election in Oriented Sparse Graphs

In [28], Santoro discusses the impact of topological information on message complexity. Results for oriented rings and oriented complete graphs are cited as evidence that additional topological information can affect the inherent message complexity of a problem. The oriented ring and the oriented complete network can be viewed as the two extremes of a class of graphs with edges labelled in a globally consistent way. In the oriented ring there is a globally consistent sense of left and right. In the oriented complete network this sense of direction is extended; local link number l connects a processor directly to the processor at distance l via the ring links. Attiya, Santoro and Zaks [9] observed that this transition from an oriented ring to an oriented complete graph results in a drop of message complexity from $\Theta(n \log n)$ to $\Theta(n)$. They also show that the $O(n)$ complexity can in fact be achieved in a oriented graph that is much sparser than the complete graph. Let $\mathcal{H}$ be a network consisting of a ring $\pi_1, \ldots, \pi_n$ augmented with the chords (π_i, π_{i+j}) and (π_i, π_{i+n-j}) for $2 \leq j \leq t-1$ at each node π_i. Then a leader for $\mathcal{H}$ can be elected in $2\frac{n}{t}\log n + 3n$ messages. Thus, there is an oriented graph consisting of a ring augmented with $\log n$ chords at each node, such that a leader can be elected in $O(n)$ messages [9].

The randomized algorithm, A on page 32, for leader election on an oriented complete graph can also be adapted to a much sparser graph without significantly increasing the expected bit complexity. Let $\mathcal{G}$ be a network of n processors $\pi_1, \ldots, \pi_n$ such that π_i is connected to π_{i+2^k} via a link with label $k, k = 0, \ldots, \lfloor \log n \rfloor$. $\mathcal{G}$ is an oriented graph with degree $\lfloor \log n \rfloor + 1$. For any d, a message can be sent between π_i and π_{i+d} using at most $\lceil \log d \rceil$ hops. By including a more elaborate *send* and *receive* structure which allows for forwarding, algorithm A can be converted to an algorithm which only uses the edges of $\mathcal{G}$. The messages need only be augmented with one additional field, *remaining distance*, leaving the other fields of the original message unchanged. Consider the *send* instruction in

line 1 of algorithm A. A message is sent directly from its source, say π_s, to its destination, say π_d, on link number *sendlink*. To send it to π_d using edges of $\mathcal{G}$, it is sent instead on link labelled a where $2^a \leq$ *sendlink* $< 2^{a+1}$, and the *remaining distance* field is set to *sendlink*-2^a. Any processor (necessarily a passive one) receiving a message with *remaining distance* not equal to 0, knows that the message is not destined for it. Suppose *remaining distance* $= d$, where $2^b \leq d < 2^{b+1}$. The forwarding processor changes the *remaining distance* field to $d - 2^b$ and forwards the message on its link labelled b. A similar strategy can be used to send the messages of lines 4 and 9 which travel in the opposite direction. The required forwarding direction can be resolved by adopting the convention that *remaining distance* is positive for messages sent in line 1 and negative for those in lines 4 and 9.

Complexity: In round j, successive active processors π_i and π_l satisfy $|i - l| \leq 2^{j-1}$. Therefore, the *remaining distance* field can always be encoded in j bits. So a total of fewer that $4(j+1)$ bits are sent by each active processor in round j. Each message requires at most $\lceil \log 2^j \rceil = j$ hops to reach its destination.

If X_j is the number of active processors in round j, the expected bit complexity is bounded above by:

$$\mathrm{E}\left(\sum_j X_j \cdot 4(j+1)j\right) = \mathrm{O}(n) \text{ since } \mathrm{E}(X_j) = \left(\frac{3}{4}\right)^{j-1} n$$

Theorem 3.6 *There is a oriented graph with n processors and $n\lceil \log n \rceil$ links on which a leader can be elected in $\mathrm{O}(n)$ expected bits by a Las Vegas algorithm.*

Chapter 4

A General Model for Asynchronous Computations on Rings

Recall that an asynchronous distributed network is modelled as a graph with nodes representing processors and edges representing communication links. As described in chapter 1, within this general model a number of additional assumptions about the network and its algorithms could be adopted. These include specification of the topology of the network, the presence or absence of identifiers, the kind of algorithm being modelled, the class of networks for which the algorithm is required to work, and input values and termination requirements of the algorithm. These various assumptions can be accommodated by a general framework which encompasses most asynchronous distributed computation. This chapter outlines this framework for general graphs and fills in the details for a restricted case. The result is a formal model of computation on an anonymous unidirectional asynchronous ring.

A distributed algorithm is modelled as an assignment of processes to the nodes of a graph. Specific characteristics of the network are captured in two ways: by specifying a collection of processes available for assignment and by varying the way in which the assignment is made. Besides providing a unified view of different distributed computing

models, this approach provides a natural and useful description of a distributed system, which facilitates the study of lower bounds. One goal of this research is to demonstrate that seemingly small changes in the details of a model for distributed computation on rings can have significant impact on the resulting communication complexity. For this reason it is necessary to have a precise model of computation that captures these details.

The overall strategy is to develop a model in two stages. First, the activity of a single node is modelled as a *process*. Since the objective is to investigate communication complexity, a process is modelled so as to highlight communication events while masking all internal processing. Information about the topology of the network and the kind of termination required of an algorithm influences the model at this level. Depending on the kind of algorithm to be modelled, either a process's communication is completely determined by its history or its communication is determined by its history and the outcome of random experiments.

The second stage is to model the way in which processes are assigned to nodes. Properties of a node such as its input value, identifier, and degree are used to restrict the collection of processes that are available to the node. For each node, a process is selected from the collection that is available to it. The way the selection is made is determined by the kind of algorithm being modelled.

By "kind of algorithm" is meant, in part, whether the algorithm is permitted to incorporate random or nondeterministic choices. In the traditional domain of sequential processing, deterministic, randomized and nondeterministic computation form a sequence of models of computation with increasing power. A similar collection of models for distributed computing can be considered. Loosely, the usual automata-theoretic notions of determinism, randomization and nondeterminism apply. It is required of a nondeterministic algorithm only that "nothing bad happens" (that is no wrong answers are produced), whereas, of a randomized algorithm, it is required that with high probability (Monte Carlo) or with certainty (Las Vegas) "eventually something good happens" (that is, right answers are produced) and a deterministic algorithm must have something good happen always. For a fixed input and scheduler, complexity of a nondeterministic algorithm is measured as

the best case and complexity of a randomized algorithm is measured as the expected case over computations producing correct answers. For a deterministic algorithm, complexity under a fixed input and scheduler is measured as the cost of the only computation that can ensue.

The processes described in this chapter are intended to reflect computation on a unidirectional ring, and hence are restricted to processes with one input communication channel and one output communication channel. Once a process is defined, the notion of a sequence of processes, and the computation arising from a ring of processes, follows naturally.

It is intended to study the relative power of deterministic and randomized algorithms that evaluate functions on an asynchronous ring. Therefore, the definitions for the correctness and complexity of function evaluation are adjusted to reflect the distinctions between deterministic algorithms versus those that incorporate randomization, and algorithms that are correct with certainty versus those that are correct with high probability. Two further classes of algorithms are also defined that generalize from purely probabilistic models to nondeterministic ones.

4.1 Processes for Rings

A *message* is an element of $M = \{0,1\}^* \cdot \Box$. The symbol $\Box$ is called the end-of-message marker. A *history* is a sequence in $\{0,1,\Box\}^*$. Then any history has a unique parse into a sequence of messages. If h is a history, then $|h|$ denotes the number of messages in h, and $\|h\|$ denotes the length of the binary encoding of h using some fixed encoding scheme to encode each symbol in $\{0,1,\Box\}$.

In a sequential processing environment, the computation of a deterministic algorithm is completely determined by its input. This property is lost only by introducing a more powerful algorithm — one that incorporates random or nondeterministic choices. In contrast, the behaviour, even on a unidirectional ring, of a non-message-driven deterministic algorithm may be determined in part by the scheduler. Various computations may arise from a fixed algorithm running on a fixed input, depending on the scheduling of message

delivery. For example, a processor may have a construct to check its input queue and follow different execution paths depending on whether or not there is a message present. Note, however, that this source of undetermined behaviour is distinct from the nondeterministic behaviour due to arbitrary choices made by the processor itself. The first source of variation in potential computation can arise even though the behaviour of each processor is entirely deterministic. That is, the processor's actions are completely determined by all the information available to it. The second source of undetermined behaviour arises only by introducing a more powerful notion of a process — one that can make random or nondeterministic choices. The following definition of a deterministic process is intended to capture the different possible behaviours of a process on a unidirectional ring that are due to the local impact of a scheduler while at the same time reflecting the fact that the process behaves completely deterministically.

A *deterministic process*, π, is a (possibly infinite) tree that models all the possible behaviours of the process. Levels of the tree alternate between *send-nodes* and *receive-nodes* with a send-node as the root. A receive-node, ρ, represents the state of π immediately after receiving, but not processing, a message. For every possible message, r, that might be received in state ρ, there is a unique directed edge labelled by r from ρ to a send-node, say σ. The node σ represents the state of π after processing message r in state ρ. In general π may send any number of messages before receiving its next message, depending on the action induced by the scheduler. Edges leaving σ represent all possible behaviours of π before π receives its next message. For each possible sequence of messages, $t = s_1, s_2, \ldots, s_k$, that may be sent by π starting from state σ before receiving another message, there is a unique directed edge labelled by t from σ to a new receive-node. Since the behaviour of a deterministic process may be determined by local information such as the state of the message queue, a process may send an unbounded number of messages before receiving another message. Hence, the branching factor of a send-node may be infinite.

For the remainder of this section it will be assumed that communication on an asynchronous unidirectional ring is message-driven with at most one message sent in response to the receipt of a message. The model as developed will then only reflect message-driven

algorithms. The relationships between general algorithms and those that are constrained to be message-driven are examined in section 4.4. As a consequence of that section, it will be possible to draw conclusions about lower bounds for general nondeterministic algorithms, from lower bounds for the restricted class of message-driven nondeterministic algorithms.

If a process is message-driven, then all state changes and output messages, except the initial transitions of initiators, are triggered by the arrival of a message. A *message-driven deterministic process* is a deterministic process such that all send-nodes of the process have branching factor 1, and the label on each edge from a send-node to a receive-node is a sequence of messages of length 0 or 1. The null message (or absence of a message) is denoted by λ. It should not be confused with $\Box$ which is the message containing only an end-of-message marker.

Process π is an *initiator* if the root of π (a send-node) has an emanating edge labelled by a message, and a *non-initiator* if it is labelled by λ.

Let $\pi_1, \ldots, \pi_n$ be a ring of message-driven deterministic processes. The notation $\pi_{1,n}$ abbreviates $\pi_1, \ldots, \pi_n$. There is a unique cyclic sequence of histories, $C = h_1, \cdots, h_n$, called a *computation* associated with $\pi_{1,n}$ in the following natural way. Each history h_i is composed of a (not necessarily finite) sequence of messages $m_{i,1} \cdots m_{i,r_i}$. If π_i is an initiator, then $m_{i,1}$ is the message labelling the edge leaving the root of π_i. The computation is then determined inductively by applying the transitions defined by π_1 through π_n and letting successive non-null output messages of π_i be successive input messages of π_{i+1}. [1]

A random process is defined by generalizing the definition of a deterministic process. In the interval between receiving two messages, a random process may make any number of random experiments. A sequence of experiments, however, may be simulated by one large experiment that produces a potentially infinite number of outcomes. Therefore, the model assumes, without loss of generality, that a random process makes one random experiment in any interval between receiving two messages. A *random process*, π, is a (possibly infinite) tree with three types of nodes. Levels of the tree cycle between send-nodes, receive-nodes

[1] It is assumed throughout that indices are reduced modulo n, and that indices n and 0 are used interchangeably.

and *choice-nodes* with a choice-node as the root. The send-nodes and receive-nodes are defined similarly to these nodes in the model of a deterministic process. A receive-node, ρ, represents the state of π immediately after receiving, but not processing, a message. For every possible message, r, that might be received in state ρ, there is a unique directed edge labelled by r from ρ to a choice-node, say χ. The node χ represents the state of π after processing message r in state ρ. For every possible outcome, o, of the random experiment done by π in state χ, there is one edge, labelled by the probability of o, from χ to a send-node, say σ. The node σ represents the state of π after an outcome of o for the random experiment of node χ. Edges leaving σ represent all possible next behaviours of π from state σ, before π receives its next message. For each possible sequence of messages, $t = s_1, s_2, \ldots, s_k$, that may be sent by π starting from state σ before receiving another message, there is a unique directed edge labelled by t from σ to a new receive-node.

The definition of a message-driven random process involves a similar restriction to that used to define a message-driven deterministic processes. A *message-driven random process* is a process such that all send-nodes of the process have branching factor 1, and the label on each edge from a send-node to a receive-node is a sequence of messages of length 0 or 1.

A message-driven random process π is an *initiator* if there exists an edge with a nonzero probability label from the root (a choice-node) to a send-node, σ, such that the edge emanating from σ is labelled by a (non-null) message. Otherwise π is a *non-initiator.*

A ring, $\pi_{1,n} = \pi_1, \ldots, \pi_n$, of independent message-driven random processes gives rise to a probability space, $\mathcal{C}$, of computations associated with $\pi_{1,n}$. The probability, $\Pr(C)$, of a computation, $C \in \mathcal{C}$, is the sum over all ways that $\pi_{1,n}$ can produce C of the product of the probabilities of the outcomes of the independent random experiments that were made for the duration of the computation.

The *bit complexity* of a computation $C = h_1, \cdots, h_n$, denoted $\|C\|$, is $\sum_{i=1}^{n} \|h_i\|$. The *message complexity* of a computation $C = h_1, \cdots, h_n$, denoted $|C|$, is $\sum_{i=1}^{n} |h_i|$. The *bit (message) complexity* of a ring, $\pi_{1,n}$, of deterministic processes is the bit (message) complexity of the computation associated with $\pi_{1,n}$. The *expected bit complexity* (respectively,

expected message complexity) *of the space of computations* associated with a ring, $\pi_{1,n}$, of message-driven random processes is $\sum_{C \in \mathcal{C}} \Pr(C) \cdot \|C\|$ (respectively, $\sum_{C \in \mathcal{C}} \Pr(C) \cdot |C|$).

A distinguished subset $M_a \subseteq M$ of messages are called *accepting* messages, and a subset $M_r \subseteq M - M_a$ are called *rejecting* messages. A history is an *accepting* history (respectively, *rejecting* history) if and only if its *last* message is an accepting message (respectively, rejecting message). A computation $h_1, \cdots, h_n$ of $\pi_1, \ldots, \pi_n$ is *accepting* if *every* h_i is an accepting history, and is *rejecting* if *every* h_i is a rejecting history. An accepting or rejecting message (respectively, history or computation) is a *decisive* message (respectively, history or computation).

Recall that for distributed termination, processes must reach irreversible conclusions. This is modelled by insisting that processes never output another message after sending a decisive message. Nondistributive termination permits a process to reach a tentative decision which it may revoke upon receipt of another message. This weaker form of termination is captured by permitting processes to follow decisive messages with messages that are not decisive or that carry the opposite decision. The final decision of the process is only determined after all message traffic has ceased — a situation that may not be detectable.

The stronger notion of distributive termination is the main concern in the context of function evaluation on a ring. To achieve interesting (super-linear) lower bounds for general nonconstant function evaluation, distributive termination must be required since, for example, nondistributive termination permits a trivial deterministic algorithm for AND which uses only $O(n)$ bits (see chapter 6, page 78).

4.2 Algorithms

Each processor in a distributed network may possess characteristics that help distinguish it from other processors in the network. Such characteristics include the processor's identifier, its degree in the network, and its input. Typically, the computation of each processor during the execution of an algorithm, is determined in part by any of these distinguishing characteristics that exist. The model captures this situation by using the characteristics

of a node to constrain the collection of processes that can be assigned to the node. Define the *label* of a node to be an encoding of all the relevant information that helps identify the node. Let labels be taken from a set D. It is usual to imagine that the node label completely determines the process that is assigned to the node. In this case an algorithm is a mapping from labels to processes. This notion can be naturally generalized to allow a mapping from labels to probability spaces of processes. The process assigned to a node is randomly selected from the space of processes available to the label of that node. A further generalization allows a mapping from labels to nonempty sets of processes. In this most general model, the process assigned to a node is arbitrarily selected from the set of processes available to the label of that node. Thus, the assignment of processes to labels can be deterministic, random, or nondeterministic. In each case the processes themselves may be deterministic or random. The relationship between these six possible combinations is examined in what follows.

4.2.1 Types of algorithms — general concepts

Section 4.1 defined processes for unidirectional rings only. In order to describe the general concepts for a collection of models for asynchronous distributed algorithms, free from the details specific to unidirectional rings, suppose there has been developed some appropriate definitions of processes for nodes in an arbitrary network. Let $\mathcal{A}$ be the set of all such processes whose next behaviour is entirely determined by the current history of the process, and $\mathcal{B}$ be the set of all processes whose next behaviour is determined by the history of the process and the outcome of a random experiment made by the process. The ideas developed in this subsection are converted to precise definitions in subsection 4.2.2 for the restricted case of computations on unidirectional rings.

If a network is executing a deterministic algorithm, the start state of each processor of the network is completely determined by its input together with any other distinguishing characteristics of the processor. Furthermore, the next state and output message of each processor is completely determined by its current input message and its history so far. A deterministic assignment of deterministic processes to labels captures this situation. Hence,

a *deterministic (distributed) algorithm*, α, is a mapping from D into $\mathcal{A}$.

In the natural description of a randomized algorithm, the start state of each node is still completely determined by its input together with any other distinguishing characteristics of the node, but random choices occur throughout the run of the algorithm. This corresponds to a deterministic assignment of random processes to labels; that is, a mapping from D into $\mathcal{B}$.

A random process, however, can be decomposed into a (possibly infinite) collection of deterministic processes each one arising from a particular outcome of a sequence of random experiments. So a random process can be viewed as a probability measure on a set of deterministic processes. (This perspective corresponds to the simulation of the choices made by a random process during the course of the algorithm's computation as a single random choice by each processor at the beginning of the computation. Essentially, the processor pre-selects all the outcomes of its random experiments.) Thus, in an alternative description of a randomized algorithm, the start state of each node is determined by its input, any other distinguishing characteristics of the node, and the result of a random experiment, and thereafter, the processor proceeds deterministically. This corresponds to a random assignment of deterministic processes to labels. Hence, a *randomized (distributed) algorithm*, α, is defined as a mapping from D to the set of probability measures on $\mathcal{A}$. The process assigned to a node with label i is chosen from the space $\alpha(i)$ according to the dictates of the probability distribution. Note that a random assignment of random processes to labels adds no additional power over the randomized algorithms just described.

A still more powerful model of distributed algorithms results if the random assignment of processes to nodes is replaced by an arbitrary one. A *nondeterministic (distributed) algorithm*, α, is a mapping from D to nonempty subsets of $\mathcal{A}$. The set $\alpha(i)$ is the collection of processes available to nodes with label i. A *nondeterministic/probabilistic (distributed) algorithm* is a mapping from D to nonempty subsets of $\mathcal{B}$. As will be seen, nondeterministic/probabilistic algorithms are more general than purely nondeterministic ones because it is possible to define a high probability of correctness in the nondeterministic/probabilistic model while this is not possible in the nondeterministic model.

In an asynchronous network, it is reasonable to assume that nodes have no control over input values or the scheduling of message delivery. These assumptions are reflected in the definitions of correctness and complexity, which are, in general, based on the worst case behaviour of algorithms over inputs and schedules. Hence, a deterministic algorithm is required to be correct for all inputs and schedules, and its complexity is the worst case over all inputs assuming an adversarial scheduler.

If the algorithm is randomized, then both correctness and complexity of the computation may depend on the outcome of random experiments. There are two types of randomized distributed algorithms. A *Las Vegas algorithm* is a randomized algorithm that terminates with probability 1 and upon termination is always correct. A more lenient notion of correctness permits erroneous results to occur with low probability. A *Monte Carlo algorithm* is a randomized algorithm that terminates with the correct result with probability at least $1-\epsilon$, where ϵ is a parameter of the algorithm with value $0 \leq \epsilon < 1$. The correctness criterion must hold for all inputs and schedules, and the complexity is defined as the worst case over inputs and schedules, and the expected case over the probability space of correct computations.

A nondeterministic algorithm also incorporates choices and different computations may ensue from a fixed input. Like conventional nondeterministic algorithms, the correctness criterion is weakened to a requirement to avoid giving wrong answers. A computation is correct if all processors terminate with the correct answer. It is erroneous if any processor terminates with the wrong answer. Notice that a computation can fail to be correct without being erroneous; for example, a computation could deadlock, or fail to terminate, or terminate with some or all processors in an undecided state. It is required of nondeterministic algorithms that none of the computations are erroneous. Like conventional nondeterministic algorithms, a nondeterministic distributed algorithm is understood to be efficient for a fixed input if there is *some* computation that yields the correct answer for that input and that has low complexity. Therefore, lower bounds on complexity in the nondeterministic model address the complexity of the *best case* over all choices and reflect the cost of certifying a correct answer. The complexity of a nondeterministic algorithm is

defined to be the worst case over inputs.

A nondeterministic/probabilistic algorithm incorporates both nondeterministic and random choices. Both correctness and complexity may depend on the outcome of these choices. It is required that for all possible nondeterministic choices, the resulting random computation yields an erroneous answer with probability less than ϵ where ϵ is a parameter of the algorithm. A nondeterministic/probabilistic algorithm is efficient for a fixed input if there are *some* nondeterministic choices for that input such that the ensuing random computation is correct with probability $1 - \epsilon$ and the correct computations that arise from these choices have low expected complexity. The complexity of such an algorithm is the worst case over inputs and schedules. If no possibility of error is to be tolerated, there is no advantage to making random choices when nondeterministic ones are permitted. In this case nondeterminism subsumes randomization. But it can be advantageous for an algorithm that errs with low probability to incorporate randomization in addition to nondeterminism, since *no* nondeterministic choices are permitted to lead to an erroneous answer, while *a few* random choices may do so.

4.2.2 Types of algorithms — definitions for rings

A collection of precise models for message-driven algorithms for unidirectional rings are constructed by refining the concepts of the previous section for this restricted case. The definitions of correctness and complexity of function evaluation on rings are achieved by specializing the ideas in subsection 4.2.1 and incorporating the definitions (section 4.1) of a unidirectional message-driven process.

Correctness

Because nodes of anonymous rings all have the same degree and lack identifiers, the only node information that can influence the assignment of processes to nodes is the input value of the node. Let input values be taken from domain D. Let f be a cyclic function from D^n to $\{0,1\}$. Then f is the characteristic function of some language $L \subseteq D^n$. Let

$I = i_1, \cdots, i_n \in D^n$. If $\pi_{1,n} = \pi_1, \ldots, \pi_n$ is a ring of message-driven deterministic processes and C is the computation associated with $\pi_{1,n}$, then the correctness of $\pi_{1,n}$ for function f on input I is captured by:

1. $\pi_{1,n}$ *strongly evaluates* f *on input* I if:

   (a) $I \in L$ implies C is an accepting computation, and

   (b) $I \notin L$ implies C is a rejecting computation.

2. $\pi_{1,n}$ *weakly evaluates* f *on input* I if:

   (a) $I \in L$ implies no history of C is a rejecting history, and

   (b) $I \notin L$ implies no history of C is an accepting history.

Similarly, if $\pi_{1,n} = \pi_1, \ldots, \pi_n$ is a ring of message-driven random processes and C is a random computation associated with $\pi_{1,n}$, then the correctness of $\pi_{1,n}$ for function f on input I is captured by:

1. $\pi_{1,n}$ *strongly evaluates* f *on input* I *with confidence* $1 - \epsilon$ if:

   (a) $I \in L$ implies $\Pr(C \text{ is an accepting computation}) \geq 1 - \epsilon$, and

   (b) $I \notin L$ implies $\Pr(C \text{ is a rejecting computation}) \geq 1 - \epsilon$.

2. $\pi_{1,n}$ *weakly evaluates* f *on input* I *with confidence* $1 - \epsilon$ if:

   (a) $I \in L$ implies $\Pr(C \text{ does not contain a rejecting history}) \geq 1 - \epsilon$, and

   (b) $I \notin L$ implies $\Pr(C \text{ does not contain an accepting history }) \geq 1 - \epsilon$.

Correctness of an algorithm is now defined in terms of the correctness of the rings of processes that can be generated by the algorithm. The strong versions of the preceding definitions are used to define randomized and deterministic message-driven algorithms that must return correct answers either with certainty or high probability. The weak versions capture the less stringent requirement of not returning an erroneous answer. In particular, a deadlocking computation meets the requirements of weak evaluation. Weak evaluation

is used to define nondeterministic and nondeterministic/probabilistic message-driven algorithms. Let $\mathcal{A}$ now denote the set of message-driven deterministic processes for unidirectional rings, and $\mathcal{B}$ denote the set of message-driven random processes for unidirectional rings. Recall that the different types of algorithms are interpreted as mappings on the domain D of labels, as follows:

A deterministic algorithm maps D to $\mathcal{A}$.

A Las Vegas or Monte Carlo algorithm maps D to the set of probability measures on $\mathcal{A}$.

A nondeterministic algorithm maps D to the set of nonempty subsets of $\mathcal{A}$.

A nondeterministic/probabilistic algorithm maps D to the set of nonempty subsets of $\mathcal{B}$.

A deterministic algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ if $\pi_{1,n} = \pi_1, \ldots, \pi_n$ strongly evaluates f on input I where $\pi_j = \alpha(i_j)$.

A Las Vegas algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ if, with probability 1, the computation of $\pi_{1,n} = \pi_1, \ldots, \pi_n$, where π_j is a random element of $\alpha(i_j)$ for $1 \leq j \leq n$, terminates, and every such $\pi_{1,n}$ strongly evaluates f on input I given that its computation terminates.

A nondeterministic algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ if, for every $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$, $\pi_{1,n}$ weakly evaluates f on input I.

A Monte Carlo algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ *with confidence* $1 - \epsilon$ if, with probability at least $1 - \epsilon$, $\pi_{1,n} = \pi_1, \ldots, \pi_n$, where π_j is a random element of $\alpha(i_j)$ for $1 \leq j \leq n$, strongly evaluates f on input I.

A nondeterministic/probabilistic algorithm α *computes* f *on input* I *with confidence* $1 - \epsilon$ if, for every $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$, $\pi_{1,n}$ weakly evaluates f on input I with confidence $1 - \epsilon$.

(The Las Vegas and Monte Carlo models also have characterizations as mappings from D to $\mathcal{B}$, which would give rise to alternative definitions of correctness. Recall that the

term *eventually P* is used to abbreviate the phrase "terminates with probability 1 and upon termination P".

A Las Vegas algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ if $\pi_{1,n} = \pi_1, \ldots, \pi_n$, where $\pi_j = \alpha(i_j)$, eventually strongly evaluates f on input I.

A Monte Carlo algorithm α *computes* f *on input* $I = i_1, \cdots, i_n$ *with confidence* $1 - \epsilon$ if, $\pi_{1,n} = \pi_1, \ldots, \pi_n$, where $\pi_j = \alpha(i_j)$, strongly evaluates f on input I with confidence $1 - \epsilon$.

Though these definitions of correctness are equivalent to those given above, they are not used here.)

An algorithm α *computes* f (*with confidence* $1 - \epsilon$) on rings of size n if it computes f on input I (with confidence $1 - \epsilon$) for every $I \in D^n$.

Functions are generally considered to have a fixed number of arguments. However, it is sometimes meaningful to consider particular functions such as AND with an arbitrary number of inputs. The definition of function evaluation is extended to include this situation, when such an extension to a variable number of inputs is warranted. Let $[a, b]$ denote a closed interval of positive integers and let $\mathcal{R}_{[a,b]}$ denote the class of all anonymous rings of size n where $n \in [a, b]$. An algorithm α *computes* f *for rings in* $\mathcal{R}_{[a,b]}$ (*with confidence* $1 - \epsilon$) if α computes f on rings of size n (with confidence $1 - \epsilon$) for every $n \in [a, b]$.

Complexity

The notion of complexity is adjusted depending on the type of algorithm that is being considered. Complexity is interpreted to mean expected complexity over random choices and best case complexity over nondeterministic choices. But in both cases, complexity addresses the amount of communication used in computations that are correct in the strong sense. Hence, though a nondeterministic algorithm may terminate without a decision or may fail to terminate, its complexity is the best case over those computations that do produce a correct answer. Similarly, though a Monte Carlo algorithm may err with low probability, its complexity is the expected communication used in correct computations. Note that a lower bound of B on the expectation over correct computations of a Monte

Carlo algorithm, α, that errs with probability at most ϵ, implies a lower bound of $(1-\epsilon)B$ on the expectation over all computations of α. These notions of complexity are captured by the following definitions.

The *bit (respectively, message) complexity of a deterministic algorithm α on input $I = i_1, \cdots, i_n$* is the bit (respectively, message) complexity of $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j = \alpha(i_j)$.

The *expected bit (respectively, message) complexity of a randomized algorithm α* (either Las Vegas or Monte Carlo) *on input $I = i_1, \cdots, i_n$* is the expectation, over the space of sequences $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where π_j is a random element of $\alpha(i_j)$, of the bit (respectively, message) complexity of $\pi_{1,n}$, given that $\pi_{1,n}$ strongly evaluates $f(I)$. (Note that sequences $\pi_{1,n}$ where $\pi_j \in \alpha(i_j)$, of a Las Vegas algorithm for function f, strongly evaluate $f(I)$ with probability 1).

The *bit (respectively, message) complexity of a nondeterministic algorithm α on input $I = i_1, \cdots, i_n$* is the minimum, over the set of sequences $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ and $\pi_{1,n}$ strongly evaluates $f(I)$, of the bit (respectively, message) complexity of $\pi_{1,n}$. If no $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ strongly evaluates $f(I)$, then the complexity of α on input I is infinite.

The *expected bit (respectively, message) complexity of a nondeterministic/probabilistic algorithm α on input $I = i_1, \cdots, i_n$* is the minimum, over the set of sequences $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ and $\pi_{1,n}$ strongly evaluates $f(I)$ with confidence $1-\epsilon$, of the expected bit (respectively, message) complexity of computations of $\pi_{1,n}$ that strongly evaluate $f(I)$. If no $\pi_{1,n} = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ strongly evaluates $f(I)$ with confidence $1-\epsilon$, then the complexity of α on input I is infinite.

The *(expected) complexity of an algorithm α* on rings of size n is the maximum over all inputs $I = i_1, \cdots, i_n \in D^n$, of the (expected) complexity of α on input I.

4.3 Relationships Between Classes of Algorithms

Deterministic, Las Vegas, and nondeterministic classes of algorithms form an hierarchy of increasingly more powerful models of computation, each derived from its predecessor by generalizing the way in which a deterministic process is chosen for a label. A Las Vegas algorithm makes a probability space of deterministic processes available to each label. If the probability space for each label is restricted to exactly one deterministic process, which is chosen with certainty, then the algorithm is deterministic. Thus:

Claim 4.1 *Lower bounds on the complexity of Las Vegas algorithms for a given problem P imply lower bounds on the complexity of any deterministic algorithm for P.*

Nondeterministic algorithms result from removing the constraint of adhering to a probability distribution when assigning a process to a label, replacing it with an arbitrary assignment from a set of processes. Nondeterministic complexity is defined as the minimum while Las Vegas complexity is defined as the expected value over the set of possible assignments that produce correct answers. Thus:

Claim 4.2 *Lower bounds on the complexity of nondeterministic algorithms for a given problem P imply lower bounds on the complexity of any Las Vegas algorithm for P.*

Recall the alternative characterization of a Monte Carlo algorithm as a mapping from the set of labels to the set of random processes. This characterization illuminates a relationship between the Monte Carlo and the nondeterministic/probabilistic models that can also be interpreted as a relaxation of constraints on the way in which processes are assigned to labels. While a Monte Carlo algorithm assigns a fixed random process to a label, a nondeterministic/probabilistic algorithm assigns a set of random processes to a label. Therefore, Monte Carlo algorithms are a subset of nondeterministic/probabilistic algorithms that assign a set of size one to each label. Consequently:

Claim 4.3 *Lower bounds on the complexity of nondeterministic/probabilistic algorithms for a given problem P imply lower bounds on the complexity of any Monte Carlo algorithm*

for P.

In addition, algorithms that require correctness with certainty can be viewed as restricted versions of the corresponding algorithms that permit error with probability at most ϵ. Setting ϵ equal to zero in the definition of correctness for a Monte Carlo algorithm yields a requirement that, with probability 1, random computations of the algorithm are correct in the strong sense. The definition of correctness for Las Vegas algorithms is even slightly stronger. Las Vegas algorithms must, with probability 1, be correct in the strong sense and the only possible incorrect computations are infinite ones (which occur with probability 0). The complexity of Monte Carlo and Las Vegas algorithms are defined in the same way — as the expected complexity over the space of strongly correct computations. Thus:

Claim 4.4 *Lower bounds for Las Vegas algorithms can be derived from lower bounds for Monte Carlo algorithms by letting ϵ tend to zero.*

Similarly, the purely nondeterministic model can be viewed as an even more constrained model than the nondeterministic/probabilistic model with ϵ set to zero. Thus:

Claim 4.5 *Lower bounds for nondeterministic algorithms can be derived from lower bounds for nondeterministic/probabilistic algorithms by letting ϵ tend to zero.*

These relationships between models are exploited in the lower bounds that are presented in chapters 5, 6 and 7. In some cases the nondeterministic model suffices to yield tight lower bounds for Las Vegas algorithms. Frequently, proving lower bounds that incorporate arbitrary distributions of random numbers is difficult. Since the nondeterministic model dispenses with probability distributions, this difficulty is finessed and the proofs become simpler.

In addition, the nondeterministic lower bounds make stronger statements than the corresponding statements in a randomized model. There are a number of interpretations of this nondeterminism in the distributed setting. First, nondeterministic lower bounds apply to algorithms that on occasion may deadlock or fail to terminate. Second, they

imply *best case* lower bounds for Las Vegas algorithms, and hence imply that even the most efficient execution of a Las Vegas algorithm has the claimed complexity. For example, the nondeterministic function evaluation lower bound in chapter 5 says that if a decisive computation (of any non-zero probability) exists for every input, then there is some input for which every decisive computation (no matter how improbable) has at least the claimed complexity.

The lower bound results herein are stated for anonymous rings; but, by a third interpretation of nondeterminism, the nondeterministic lower bounds apply in a more general setting. One might imagine an algorithm that works on all rings, and that is especially efficient on a ring with processors named in a particular way. Since nondeterministic choices can be interpreted as choices of names for processors, the nondeterministic lower bounds preclude such algorithms.

Finally, as will be seen in section 4.4, lower bounds for message-driven nondeterministic algorithms extend to lower bounds for even the best possible scheduling of (non-message-driven) nondeterministic algorithms.

There are some problems for which the nondeterministic model is too powerful to admit tight lower bounds for randomized algorithms. In some cases, strong Las Vegas lower bounds cannot be constructed from the tools for proving nondeterministic lower bounds only because deadlock in permissible in the nondeterministic model but not in the Las Vegas model. For such problems, constraining the nondeterministic model slightly by admitting only nondeadlocking algorithms is sufficient to achieve the desired Las Vegas lower bounds. The addition of this constraint to nondeterminism models the best case execution of a Las Vegas algorithm. For other problems, this approach is not adequate. Attrition (chapter 2, page 13) on rings of known size is one such problem. A randomized procedure for attrition, which is very efficient in the best case, is described in chapter 7. However, the expected complexity of attrition exceeds its best case complexity. A general lower bound for the expected complexity of attrition that deadlocks with probability at most ϵ is derived in chapter 7. The relationship between the Las Vegas and the Monte Carlo models is then exploited to get a lower bound for nondeadlocking attrition as a corollary.

The proofs that depend on the nondeterministic model all have some structural similarity. The existence of some correct computation with low bit complexity is assumed. This computation is then manipulated to form a new computation which leaves one or more processors in an erroneous state. Since the new computation is a possible computation of a ring in the class for which the algorithm is supposed to work, it is concluded that the assumption of low complexity must be incorrect.

There is a collection of techniques for carrying out the manipulations of rings of processes. Because few bits were expended during the computation, it follows that there are repeated histories. *Shrinking* removes sections between repeated histories while ensuring that the resulting sequence of histories remains a possible computation of the algorithm on an appropriately smaller ring. *Replication* concatenates many copies of the new computation producing a possible computation of the algorithm on an enlarged ring. When necessary, *splicing* is used to construct a computation on a ring meeting an exact size requirement. This technique inserts a new segment into a computation while ensuring that at least one history in the resulting computation remains unaffected by the insertion. Lemmas providing specific tools to shrink, replicate and splice are introduced in the following chapters as needed.

The techniques used to derive nondeterministic lower bounds have been enriched to work in the nondeterministic/probabilistic model (see [3,4]) but they will not be developed in detail here. The tools for the nondeterministic/probabilistic model are considerably more complex than the corresponding purely nondeterministic ones. In the nondeterministic/probabilistic model, the initial assumption of low expected complexity applies to the whole ring rather than to a specific computation. In addition, the initial correctness assumption has an attached probability. Therefore, it is necessary to manipulate the whole probability space of computations that may arise and to account for the associated probabilities. Since each application of a ring manipulation reduces the probability of the resulting set of computations, the proofs must be frugal in their use of shrinking, replicating and splicing. As a consequence, the machinery of these proofs is much more elaborate than that required in the purely nondeterministic domain.

It is sometimes helpful in lower bound proofs to describe computations by focusing on messages rather than on processors. In a message-driven computation, if a processor responds to the receipt of a message by sending on a new one, then, potentially, the new message contains information that was sent by the recipient's predecessor. This propagation of information from one processor to another is captured by the concept of *message envelopes*. Each initiator creates an envelope, inserts its initial message into the envelope and sends it to its successor. Upon receipt of an envelope a processor removes and processes the contained message. It then either inserts a new message and forwards the envelope, or it destroys the envelope. A computation terminates precisely when all message envelopes have been destroyed. Suppose that the envelope created by π_i is destroyed by processor π_j. Then the *trace* of this envelope is the sequence of processors $\pi_i, \ldots, \pi_j$ that had possession of the envelope at some time during the computation (cf. [23]). The perspective of envelope traces and envelope initiators and terminators aids in the description of some of the manipulations of computations in the proofs of the following chapters and is therefore frequently adopted.

4.4 Extensions to Non-message-driven Algorithms

Although the general definition of a process on page 42 incorporates non-message-driven behaviour, the details of the development of the model for unidirectional rings depend upon the restriction to message-driven processes defined on page 43. It is intended, however, to use the models to prove lower bounds that are as general as possible for distributed algorithms for rings. In particular, it is desirable that the statements of results are not restricted to message-driven algorithms. In fact, lower bounds for the bit complexity of message-driven nondeterministic algorithms for rings extend in a strong sense to lower bounds for unconstrained nondeterministic algorithms for rings, as is now shown.

Any (non-message-driven) deterministic process can be viewed as a possibly infinite collection of message-driven deterministic processes under a two step conversion. Recall from the definitions of non-message-driven and message-driven deterministic processes that

the distinction is in the constraints imposed on the edges emanating from send-nodes. A send-node of a non-message-driven process may have an unbounded branching factor, but a send-node of a message-driven deterministic process has branching factor one.

Let σ be a send-node of a deterministic process τ with branching factor greater than one. For every edge, t, directed out of σ to a receive-node, say ρ, there is a process, τ_t, created from τ by removing all subtrees of σ except that rooted at ρ. Let $\tau(\sigma)$ be the set of all such processes τ_t formed from τ where t is an edge directed out of σ. The processes in the set $\tau(\sigma)$ each describe one possible behaviour of τ from state σ. Imagine that the construction is now reiterated for every process $\tau' \in \tau(\sigma)$ that has branching factor greater than one at some send-node, σ'. If this decomposition is continued until all send-nodes have branching factor one, the result is a (possibly infinite) set, $\mathcal{L}(\tau)$, of processes, all of which have send-nodes with branching factor one. For each process $\gamma \in \mathcal{L}(\tau)$, the edge in γ from any send-node, σ, to its succeeding receive-node, ρ, represents a transition of τ from state σ to state ρ that is possible under some action of the scheduler.

Consider a computation, $C_{\tau,S}$, of a ring of non-message-driven deterministic processes, $\tau = \tau_1, \ldots, \tau_n$, under any scheduler, S. For each $1 \leq i \leq n$, there is some element $\gamma_i \in \mathcal{L}(\tau_i)$ such that the computation, C_γ, of $\gamma = \gamma_1, \ldots, \gamma_n$ is the same as $C_{\tau,S}$. Now consider a sequence $\gamma = \gamma_1, \ldots, \gamma_n$ where $\gamma_i \in \mathcal{L}(\tau_i)$. Because of the definition of asynchrony, any possible action of a process from a state corresponding to a send-node can be forced by some action of the scheduler. As a consequence, for every $\gamma = \gamma_1, \ldots, \gamma_n$ where $\gamma_i \in \mathcal{L}(\tau_i)$, there is a scheduler S' such that the computation $C_{\tau,S'}$ of $\tau = \tau_1, \ldots, \tau_n$ is the same as the computation of γ.

The set $\mathcal{L}(\tau)$ is not a set of message-driven processes because the edges emanating from a send-node may be labelled with a sequence containing more than one message. That is, more than one message may be sent in response to a message received. However, a specification of a process that describes its behaviour in state ρ given a single input message, is easily interpreted as a description of the behaviour of the process in state ρ given a sequence of input messages. Informally, the process sends as output the concatenation of the sequences of output messages it would send if it sequentially responded to each message

in the input sequence separately. Sequences of messages can then be viewed as individual messages in a larger message space. This step is made formal by replacing each process $\gamma \in \mathcal{L}(\tau)$ with a message-driven one. First construct the directed acyclic graph $d(\gamma)$ from γ using the following closure operation. For every directed path p in γ from a receive-node ρ to a receive-node ρ', where ρ' is a descendant of ρ, add a new directed path of length two consisting of an edge e_1 from ρ to a new send-node, say σ, and an edge e_2 from σ to ρ'. Label edge e_1 with the concatenation of the labels of all edges leaving receive-nodes along path p. Label edge e_2 with the concatenation of the labels of all edges leaving send-nodes along path p. Next, transform the directed acyclic graph $d(\gamma)$ into a tree $md(\gamma)$ in the standard way by progressing through $d(\gamma)$ from the source (originally the root of γ), and for each node ν that has indegree *deg* larger than 1 replicate *deg* times, the node ν together with the subgraph rooted at ν. The labels on edges of tree $md(\gamma)$ contain elements of M^* where M is the original message space of τ. It is a routine matter to encode sequences of messages as a single packet in a packet space consisting of elements of $M^* \cdot \Delta$, where Δ is an end-of-packet marker. Let $md'(\gamma)$ be the process $md(\gamma)$ with the edges relabelled according to this encoding. Finally, let $\mathcal{M}(\tau) = \{md'(\gamma) | \gamma \in \mathcal{L}(\tau)\}$. Then $\mathcal{M}(\tau)$ is a set of message-driven processes and each element of $\mathcal{M}(\tau)$ reflects a possible execution of τ.

The correspondence between an arbitrary deterministic process τ and the set of message-driven deterministic processes $\mathcal{M}(\tau)$ is used to extend lower bounds for message-driven nondeterministic algorithms to lower bounds for non-message-driven nondeterministic algorithms.

Lemma 4.6 *There is a constant k such that for every non-message-driven nondeterministic algorithm β there is a message-driven nondeterministic algorithm α and for every computation C_β of $\rho_1, \ldots, \rho_n$ under scheduler S where $\rho_i \in \beta(i)$, there corresponds $\pi_1, \ldots, \pi_n$ where $\pi_i \in \alpha(i)$ and the computation C_α of $\pi_1, \ldots, \pi_n$ satisfies:*

1. *The final state of π_i is the same as the final state of ρ_i under scheduler S.*
2. *The bit complexity of C_α is at most k times the bit complexity of C_β.*

Proof: Suppose β is a (not necessarily message-driven) nondeterministic algorithm for problem P. Then β can be viewed as an assignment of sets of deterministic non-message-driven processes to possible input values of processors. Define the message-driven nondeterministic algorithm α by: for every processor input i, $\alpha(i) = \bigcup_{\tau \in \beta(i)} \mathcal{M}(\tau)$. By the arguments above, the set of behaviours of a process τ is the same as the set of behaviours of processes in $\mathcal{L}(\tau)$. Furthermore each process in $\mathcal{L}(\tau)$ corresponds to one in $\mathcal{M}(\tau)$. As a consequence, for any computation C_β of a sequence $\tau_1, \ldots, \tau_n$ where $\tau_i \in \beta(i)$ there is some $\pi_1, \ldots, \pi_n$ where $\pi \in \alpha(i)$ and the computation C_α of $\pi_1, \ldots, \pi_n$ is an encoding, in the larger message space, of the computation C_β. In particular the computations leave corresponding processes in corresponding states, and the number of bits in each computation is the same, up to the cost of the encoding of sequences of messages into single messages. ■

As an immediate consequence:

Theorem 4.7 *A lower bound on the bit complexity of any message-driven nondeterministic algorithm for problem P on a ring implies a lower bound of the same order on the bit complexity of any non-message-driven nondeterministic algorithm for P on a ring even for the best possible scheduler.*

Proof: Let $\Omega(f(n))$ be a lower bound on the bit complexity of message-driven algorithms for P. Suppose β is a (not necessarily message-driven) nondeterministic algorithm for P. Define the message-driven nondeterministic algorithm α as in lemma 4.6. Since α is a message-driven algorithm for P, for some input $I \in D^n$, every decisive computation of α on I has bit complexity at least $\Omega(f(n))$. Since for each computation of β, there corresponds a computation of α with complexity within a constant factor of that of β, every computation of β on input I also has complexity $\Omega(f(n))$. ■

The preceding discussion extends lower bounds for nondeterministic message-driven algorithms to lower bounds for nondeterministic non-message-driven algorithms. The same arguments can be carried out for lower bounds on the best case execution of Las Vegas algorithms.

Corollary 4.8 *A lower bound on the best case bit complexity of any message-driven Las Vegas algorithm for problem P on a ring implies a lower bound of the same order on the best case bit complexity of any non-message-driven Las Vegas algorithm for P on a ring even for the best possible scheduler.*

Chapter 5

Minimum Nonconstant Function Evaluation

Suppose that an anonymous ring of n processors, $\pi_1, \ldots, \pi_n$, each with a single input value (say $i_1, \cdots, i_n$ respectively), must cooperate to compute $f(i_1, \cdots, i_n)$ for some fixed n-variable function f. Since computation is on a ring of identical processors, the function f is assumed to be *cyclic*, that is $f(i_1, \cdots, i_n) = f(i_j, \cdots, i_n, i_1, \cdots, i_{j-1})$ for any $1 \leq j \leq n$. The general question of the minimum communication complexity required to compute any nonconstant function f on an anonymous ring was first addressed by Moran and Warmuth in [21]. They focus on the complexity of nonconstant functions because any constant function can be evaluated without any communication. They prove that:

1. There is a nonconstant cyclic boolean function f such that f can be computed by a *deterministic* algorithm in $O(n \log n)$ bits of communication on a ring of size n.

2. If g is any nonconstant cyclic function of n variables, then any *deterministic* algorithm for computing g on an anonymous ring of size n has complexity $\Omega(n \log n)$ bits of communication.

For function evaluation by algorithms employing randomization, the lower bound does not apply and the upper bound can be improved. Specifically, this chapter shows[1] that:

1. There is a nonconstant cyclic boolean function f, such that f can be computed by a *Las Vegas* algorithm in $O(n\sqrt{\log n})$ expected bits of communication on a ring of size n.

2. If g is any nonconstant cyclic function of n variables, then any *nondeterministic* algorithm for computing g on an anonymous ring of size n has complexity $\Omega(n\sqrt{\log n})$ bits of communication.

Notice that the upper bound is achieved by a Las Vegas algorithm that evaluates a fixed function f, whereas the lower bound is a nondeterministic one. Thus, although there is a nontrivial function that can be evaluated using $O(n\sqrt{\log n})$ bits on the average even for the worst input, there is no nontrivial function that can be evaluated in less than this complexity even given the best possible outcome of random experiments. Essentially, just the verification of a guessed function value requires this minimum amount of communication for some input.

5.1 Tools for Proving Lower Bounds

A number of lemmas follow immediately from the definitions of chapter 4. They allow us to manipulate computations, building new ones from old ones.

Lemma 5.1 *If $h_1, \cdots, h_k$ is a history sequence with complexity less than b, and all h_i are distinct, then $k < \frac{3b}{\log b}$.*

Proof: At least $(k \log k)/2$ bits are required to encode k distinct strings. Hence, $(k \log k)/2 < b$ implying $k < \frac{3b}{\log b}$. ∎

[1] The content of this chapter is a modified version of material in [6].

Lemmas 5.2 and 5.3 provide, respectively, the shrinking and replicating tools that were promised in chapter 4. Each tool transforms an initial ring of processes, π, into a new ring, ρ, containing only processes that appeared in π. The proofs follow easily by considering the propagation of message envelopes. The sequence of messages received by any fixed process in ρ is identical to the sequence it received in π.

Lemma 5.2 *If $C = h_1, \cdots, h_n$ is the computation of the ring $\pi_1, \ldots, \pi_n$ of deterministic processes with $h_i = h_j$ for some $i < j$, and π_1 is the only initiator, then $C' = h_1, \cdots, h_i, h_{j+1}, \cdots, h_n$ is the computation of $\pi_1, \cdots, \pi_i, \pi_{j+1}, \cdots, \pi_n$.*

Note that it is essential that the single initiator remains in the ring, for lemma 5.2 to hold, since otherwise the computation would not even begin. In contrast, the replication of computations is entirely unconstrained; the behaviour of a process in a replicated ring is indistinguishable from its behaviour in the original ring.

Let $(x)^r$ denote the concatenation of r copies of sequence x.

Lemma 5.3 *If $C = h_1, \cdots, h_n$ is the computation of the ring $\pi_{1,n} = \pi_1, \ldots, \pi_n$ of deterministic processes then $C^r = (h_1, \cdots, h_n)^r$ is the computation of $(\pi_{1,n})^r = (\pi_1, \ldots, \pi_n)^r$.*

Using a more elaborate construction than that employed in lemma 5.3, a new ring can be created by first replicating the sequence of processes in the original ring and then splicing an additional sequence of processes into the replicated ring. Even in this more general situation, the replication ensures that a prefix of a process's history in the original ring matches a prefix of the process's history in the final ring. This is the content of the following lemma. It is helpful to have a notation for the first r messages of h, provided that h contains at least r messages. To this end, let $h|_r$ denote the sequence consisting of the first x messages of history h where $x = \min\{r, |h|\}$.

Lemma 5.4 *Let $h_1, \cdots, h_n$ be the computation of the ring $\pi_{1,n} = \pi_1, \ldots, \pi_n$ of deterministic processes and let τ be any sequence of processes. Let $g_1, \ldots, g_m$ be the computation of $\rho = (\pi_{1,n})^r \tau$. Then for $0 \leq j \leq r-1$ and $1 \leq i \leq n$, $g_{jn+i}|_j = h_i|_j$.*

Proof: The lemma is trivially true if there are no initiators in $\pi_{1,n}$ since then $|h_i| = 0$ for each i. So assume that $\pi_{1,n}$ contains at least one initiator. Let $\rho_{j,i}$ denote the $(j+1)^{st}$ copy of π_i in ρ. The lemma is clearly true for $j = 0$. Suppose it holds for $j < r - 1$. Then each process $\rho_{j,i}$ sends at least the first x messages in its history, h_i, where $x = \min\{j, |h_i|\}$. The initiators in the $(j+1)^{st}$ copy of $\pi_{1,n}$ each produce a message before processing any received messages. As a result, the computation of the $(j+2)^{nd}$ copy of $\pi_{1,n}$ in ρ agrees with the original computation of $\pi_{1,n}$ for one more round of messages. Hence, process $\rho_{j+1,i}$ sends at least the first y messages of its history h_i, where $y = \min\{j+1, |h_i|\}$. ■

5.2 Bit Complexity of Function Evaluation — Lower Bound

Let f be any nonconstant cyclic function of n variables, and let α be any nondeterministic algorithm that computes f on a ring of size n. It is shown in this section that there exists some input string I for which α requires $\Omega(n\sqrt{\log n})$ bits of communication to compute $f(I)$. Thus, the complexity of α is $\Omega(n\sqrt{\log n})$ bits.

The proof proceeds in two steps. The first step is to show that the claimed complexity applies whenever α is restricted to single initiator computations. The second step is a reduction which proves that any algorithm for function evaluation can be converted to an algorithm that works for any preassigned nonempty subset of initiators without entailing any significant additional complexity.

Lemma 5.5 *Let $f: D^n \longrightarrow \{0,1\}$ be any nonconstant cyclic decision function of n variables. Let α be any message-driven nondeterministic algorithm that computes f on a ring of size n. Then there is some input $I \in D^n$ for which α requires $\Omega(n\sqrt{\log n})$ bits of communication to evaluate $f(I)$ whenever the number of initiators in the computation of α is 1.*

Proof: The proof has two steps. First, a new computation of α is constructed from a given one that has low complexity and a single initiator. Second, an input is found such that if α has low complexity on that input, then the construction of step one would result in an erroneous new computation.

Let $I = i_1, \cdots, i_n \in D^n$ and let $C = h_1, \cdots, h_n$ be the computation of $\pi = \pi_1, \ldots, \pi_n$ where $\pi_j \in \alpha(i_j)$ and $\pi_1, \ldots, \pi_n$ has exactly one initiator. Without loss of generality, assume the single initiator is π_1. Suppose the complexity of C is less than $(n\sqrt{\log n})/3$. Let $r = |h_1|$. Because there is exactly one initiator and α is message-driven, each history has either r or $r-1$ messages. Each message requires at least one bit, therefore $n(r-1) < (n\sqrt{\log n})/3$ implying $r < \sqrt{\log n}/2$ for large enough n.

C is now collapsed by repeatedly applying lemmas 5.1 and 5.2 until all histories are distinct. Let $C' = h_{\beta_1}, \cdots, h_{\beta_l}$ be the resulting subsequence of C. Then by lemma 5.2, C' is the computation of $\pi' = \pi_{\beta_1}, \cdots, \pi_{\beta_l}$ with input $I' = i_{\beta_1}, \cdots, i_{\beta_l}$. By lemma 5.1, $l < n/\sqrt{\log n}$. By lemma 5.3, $(C')^r$ is a computation of $(\pi')^r = \pi^1_{\beta_1}, \cdots, \pi^1_{\beta_l}, \pi^2_{\beta_1}, \cdots, \pi^2_{\beta_l}, \cdots, \pi^r_{\beta_1}, \cdots, \pi^r_{\beta_l}$ with input $(I')^r$ of length $rl < n/2$.

Let $\gamma = \gamma_1, \cdots, \gamma_{n-rl}$ be any element of D^{n-rl} and $\tau = \tau_1 \cdots \tau_{n-rl}$ be any sequence such that $\tau_j \in \alpha(\gamma_j)$. Consider the sequence $(\pi')^r\tau$ which has input $(I')^r\gamma$. By lemma 5.4, the first $r-1$ messages received by $\pi^r_{\beta_1}$ in $(\pi')^r\tau$ match the first $r-1$ messages received by π_1 in π. Hence, $\pi^r_{\beta_1}$ must have the first r messages of h_1 as its first r output messages. In particular, if h_1 is decisive, h_1 is the complete history of $\pi^r_{\beta_1}$.

In summary, this construction of shrinking, replicating and splicing can be applied to any $\pi_{1,n}$ and its input I whenever the computation, C, of $\pi_{1,n}$ is decisive and has complexity less than $n(\sqrt{\log n})/3$ and $\pi_{1,n}$ has only a single initiator. The result is a new sequence $(I')^r$ of length $rl < n/2$ such that for any γ of length $n - rl > n/2$, there is a computation C'' of α on input sequence $(I')^r\gamma$, and there is a decisive history h_i occurring in both C and C''.

This construction is now used to find an input $I \in D^n$ for which the assumption that there exists a single initiator decisive computation of α on input I with complexity less than $(n\sqrt{\log n})/3$, leads to a contradiction.

Let $L \subset D^n$ be the language recognized by f. Let $d \in D$ and without loss of generality, assume $d^n \notin L$. (Otherwise consider the language $\bar{L}$.) Let $\omega = d^k\rho$ be an element of L such that k is maximum over all strings in L.

Case 1: Suppose $k \leq n/2$. Then let $I = \omega$. Suppose there is a single initiator computation C of α on I such that $\|C\| < (n\sqrt{\log n})/3$ and C strongly computes $f(I)$. Since $\omega \in L$, C must be an accepting computation. Then, by applying the construction described above, there is an sequence I'' of length $rl < n/2$ and a computation C'' of α on $I''d^{n-rl}$ containing an accepting history. This is a contradiction because the input contains at least $n - rl > n/2 \geq k$ consecutive d's and should therefore generate a rejecting computation.

Case 2: Suppose $k > n/2$. Then let $I = d^n$. Suppose there is a single initiator computation C of α on I such that $\|C\| < (n\sqrt{\log n})/3$ and C strongly computes $f(I)$. Since $d^n \notin L$, C must be a rejecting computation. Then there is an sequence $I'' = d^{rl}$ where $rl < n/2$ and a computation C'' of α on $I''d^{k-rl}\rho = d^k\rho = \omega$ containing a rejecting history. This is a contradiction because $\omega \in L$ and should therefore generate an accepting computation. ■

The bit complexity of a nondeterministic algorithm on a fixed input is defined as the best case over all possible assignments of processes that are consistent with the input and that result is a correct computation. The previous lemma only addresses the bit complexity of the best case on an input over those consistent assignments of processes that happen to have one initiator. Given a function f, let I_f^* be the input that is found in the previous lemma. It remains a possibility that the bit complexity of an algorithm α for f is less than $n(\sqrt{\log n})/3$ bits on input I_f^* for some other assignment of processes with a different number of initiators. It will now be shown that such a situation cannot arise. That is, there is no loss of generality in assuming a single initiator (or any convenient non-empty set of initiators) when communication complexity is measured in bits.

Let α be a (not necessarily message-driven) nondeterministic algorithm that computes some function f on anonymous rings of size n. For each possible input i and for each process $\pi \in \alpha(i)$ two new processes, D(π) and N(π), are constructed. D(π) consists of π augmented with two new states and edges. The root of D(π) is a send-node, say σ, with an edge labelled "wake-up" from σ to a receive-node, say ρ. An edge labelled "wake-up" is directed from ρ to the root of π. N(π) is also constructed by augmenting π with two new states and edges. The root of N(π) is a send-node, say σ', with an edge labelled with the

null message λ directed from σ' to a receive-node, say ρ'. An edge labelled "wake-up" is directed from ρ' to the root of π.

Define algorithm, $\hat{\alpha}$, by $\hat{\alpha}(i) = \bigcup_{\pi \in \alpha(i)} (D(\pi) \cup N(\pi))$. Then it is straightforward to confirm that $\hat{\alpha}$ is a nondeterministic algorithm that also computes f. For any sequence $\pi_{1,n} = \pi_1, \ldots, \pi_n$ of processes in α, there corresponds 2^n sequences of $\hat{\alpha}$. Each of these sequences that has at least one initiator, has a computation consisting of a single round of exchanged "wake-up" messages followed by the same computation as $\pi_1, \ldots, \pi_n$. Thus, each has bit complexity within $O(n)$ of the bit complexity of $\pi_1, \ldots, \pi_n$. Furthermore, one of the 2^n computations corresponds to each possible way that initiating and non-initiating status happens to be assigned to the processors on the ring.

Let $P \subseteq \{1, \ldots, n\}$ designate a set of initiators, and let $I = i_1, \cdots, i_n$ be an input sequence. Let $\text{Cost}(\alpha, I, P)$ denote the minimum, over all sequences $\pi_{1,n} = \pi_1, \ldots, \pi_n$ such that $\pi_j \in \alpha(i_j)$ and π_i is an initiator if and only if $i \in P$, of the complexity of $\pi_{1,n}$. With this notation, the conversion of α to $\hat{\alpha}$ can be summarized as follows.

Lemma 5.6 *A nondeterministic algorithm $\hat{\alpha}$ can be constructed from any (not necessarily message-driven) nondeterministic algorithm α that computes some function f on rings of size n, such that: (1)There exist computations of $\hat{\alpha}$ that compute f on rings of size n for any set of initiators, and (2) For every input I, the maximum over non-empty sets P, of $\text{Cost}(\hat{\alpha}, I, P)$, is at most the complexity of α on I plus $O(n)$.*

The combination of lemmas 5.5, 5.6 and 4.6 imply the following theorem.

Theorem 5.7 *Let f be any nonconstant cyclic decision function of n variables. Then the complexity of any nondeterministic algorithm that computes f on a ring of size n is $\Omega(n\sqrt{\log n})$ bits even assuming the best possible scheduler.*

Proof: Let α be any nondeterministic algorithm that computes f on rings of size n. By lemma 5.6, there is an algorithm $\hat{\alpha}$ that computes f on rings of size n and, for every input I, the maximum over any non-empty set P, of $\text{Cost}(\hat{\alpha}, I, P)$, is at most the complexity of α

on I plus $O(n)$. By lemma 4.6, $\hat{\alpha}$ can by converted to a message-driven algorithm β with at most a linear increase in bit complexity for every input. It follows that there is a message-driven nondeterministic algorithm β that computes f on rings of size n and for every input I, the maximum over non-empty sets P, of $\text{Cost}(\beta, I, P)$ is at most the complexity of α on I plus $O(n)$. Thus, for every input I, if $P = \{1\}$, then $\text{Cost}(\beta, I, P)$ is at most the complexity of α on I plus $O(n)$. But β is a message-driven nondeterministic algorithm that computes f on rings of size n and has a computation for any set of initiators. Therefore, by lemma 5.5 there is an input I_f^* such that if $P = \{1\}$, then $\text{Cost}(\beta, I, P) = \Omega(n\sqrt{\log n})$. It follows that α has complexity $\Omega(n\sqrt{\log n})$ bits. ■

Suppose f is any nonconstant cyclic function with range S, and $s_1 \in S$ is one possible value of f. Then the function f_1 defined by

$$f_1(x) = \begin{cases} 1 & \text{if } f(x) = s_1 \\ 0 & \text{otherwise} \end{cases}$$

is a nonconstant cyclic decision function which can be computed at least as cheaply as f. Therefore, the lower bound above actually applies to general, not just decision, functions on a ring.

5.3 A Function that Achieves Minimum Bit Complexity

This section presents a nonconstant boolean function f, which can be computed by a Las Vegas algorithm in $O(n\sqrt{\log n})$ expected bits on a ring of size n.

The algorithm for f relies on an algorithm for a simpler problem called *solitude detection*. Let a nonempty set of processors in a distributed system be distinguished. The problem of solitude detection is for every distinguished processor to determine whether there is one or more than one distinguished processor. In an algorithm for solitude detection, the initiators are precisely the distinguished processors. As will be seen, solitude detection can be solved by a Las Vegas algorithm with expected complexity $O(n\sqrt{\log n})$ bits when ring size n is fixed. (The more general Monte Carlo version of this algorithm

appears in [4]. See [3,4,5] for details on the complexity of solitude detection for a ring when various assumptions are made concerning the requirements of the solution.)

Solitude detection algorithm

Let m be the smallest integer such that $m \geq \sqrt{\log n}$ and m is relatively prime to n. It follows from the prime number theorem that $m = O(\log n)$ [22]. Messages have two fields; *message type* and *message value*. For ease of description, five message types are used. However, it is really only necessary to label *alarms* since other message types arrive in a fixed order and can thus be distinguished implicitly. The function *random*(x) returns an unbiased random coin toss (with outcome heads or tails) and stores the result in variable x.

Algorithm *SD*:

```
Initiators:
   send(<coin-toss, random(my-toss)>);
   round ←0; terminated ←false;
   WHILE NOT terminated DO
      receive(<type, value>);
      CASE type OF
         coin-toss:  IF value = my-toss THEN
                        IF round < log log n THEN
                           send(<coin-toss, random(my-toss)>);
                           round ←round+1
                        ELSE
                           send(<mod-count, 1>);
                           round ←0
                     ELSE more-than-one.
         mod-count:  IF value = n mod m THEN
                        send(<gap-count, 1>)
                     ELSE more-than-one.
         gap-count:  IF value = ''long '' THEN
                        send(<okay, -->)
                     ELSE more-than-one.
         okay:       IF round < √(log n) THEN
                        send(<okay, -->);
                        round ←round+1
                     ELSE alone ←terminated ←true.
         alarm:      more-than-one.
```

```
   PROCEDURE more-than-one:
      send(<alarm, -->); alone ←false; terminated ←true.

Non-initiators:
   REPEAT forever
      receive(<type, value>);
      CASE type OF
         coin-toss:  forward message.
         okay:       forward message.
         alarm:      forward message.
         mod-count:  send(<mod-count, (value + 1) mod m>).
         gap-count:  IF value < n/√log n THEN
                        send(<gap-count, value+1>)
                     ELSE
                        send(<gap-count, ''long''>).
```

Theorem 5.8 *Algorithm $\mathcal{SD}$ solves the solitude detection problem on unidirectional rings with expected communication complexity $O(n\sqrt{\log n})$ bits.*

Proof: The correctness and complexity proofs are outlined here; the details appear in [4].

Correctness: When there is only one initiator,then it is readily confirmed that no alarms are generated and the algorithm terminates with alone assigned true for the sole initiator.

When there are $k \geq 2$ initiators, the "mod-count" messages ensure that an alarm is generated unless $k \geq m+1$. The "gap-count" messages then ensure that at least one alarm is generated within every sequence of $\sqrt{\log n} \leq m$ initiators. Finally the "okay" messages ensure that alarms are forwarded to any initiators that have not already sent one, thus informing all initiators of nonsolitude.

Complexity: When there is one initiator, the coin tosses never produce an alarm so they account for $O(n\sqrt{\log n})$ bits. Counting modulo m requires $O(n \log m) = O(n \log\log n)$ bits. A gap counter, originating at the initiator, is incremented by each non-initiator until the counter reaches a value of $n/\sqrt{\log n}$. After that, the constant length message, "**long**", is propagated around the ring. Therefore, a further $O\left(\frac{n}{\sqrt{\log n}} \log n\right) = O(n\sqrt{\log n})$ bits are used by the gap counter. Finally, $\sqrt{\log n}$ okay messages of constant length propagate around the ring accounting for $O(n\sqrt{\log n})$ bits. Thus, the complexity, when there is one

initiator, is $O(n\sqrt{\log n})$ bits.

When there are two or more initiators, the probability is $\left(1 - \frac{1}{\log n}\right)$ that a given initiator will send an alarm before successfully sending and receiving $\log\log n$ pairs of matching coin tosses. Therefore, the total expected bit complexity of "mod-count", "gap-count", and "okay" messages is $O\left(\frac{n}{\log n}(\log m + \log n + \sqrt{\log n})\right) = O(n)$. The expected cost of the coin tosses is $O(n)$ bits since each initiator sends $O(1)$ expected bits before sending an alarm. Alarms cost $O(n)$ bits always. So the total cost is $O(n)$ expected bits, when there are two or more initiators. ■

$\mathcal{SD}$ distinguishes between one and more than one initiator. Since nothing is computed if there are no initiators, $\mathcal{SD}$ cannot be trivially converted to a boolean function over all strings in $\{0,1\}^n$. A boolean function is now constructed which, after a small amount of communication (at most $O(n\log\log n)$ bits), always leaves at least one processor in a distinguished state, and for some nonempty subset W of inputs, leaves exactly one processor in a distinguished state. The distinguished processors can then determine whether there is one or more than one distinguished processor by running $\mathcal{SD}$. Hence, the processors determine whether or not the input string is in W.

Let $\nu(n)$ be the smallest positive nondivisor of n. Note that $\nu(n) = O(\log n)$. Let $t = \lceil \log \nu(n) \rceil$ and $T = 2t + 2$. Assume $T < \nu(n)$. (Otherwise $\nu(n) \leq 10$ and a simpler approach results in a function that reduces to solitude detection in $n \cdot \nu(n) = O(n)$ bits.) Let $r = n/T$. Note that r is an integer.

A configuration of bits on a ring of size n is *well-formed* if it has the form $(1 \cdot 1 \cdot (0 \cdot \{0,1\})^t)^r$. Note that a well-formed configuration has a unique parse into r *blocks* of T bits of the form $1 \cdot 1 \cdot (0 \cdot \{0,1\})^t$. A block $1 \cdot 1 \cdot 0 \cdot b_{t-1} \cdot 0 \cdot b_{t-2} \cdot 0 \ldots 0 \cdot b_0$ *encodes* the integer whose binary representation is $b_{t-1}b_{t-2}\ldots b_0$. A well-formed configuration is *sequential* if successive blocks (including block r followed by block 1) encode successive integers mod $\nu(n)$. A well-formed configuration is *almost sequential* if all but one pair of successive blocks encode a pair of successive integers mod $\nu(n)$. Since $\nu(n)$ does not divide n, sequential configurations do not exist. However, almost sequential configurations are

easily constructed.

Let f be the boolean function defined on strings $\omega \in \{0,1\}^n$ by:

$$f(\omega) = \begin{cases} 1 & \text{if } \omega \text{ is almost sequential} \\ 0 & \text{otherwise} \end{cases}$$

Theorem 5.9 *Evaluation of f on a distributed ring reduces to solitude detection in $O(n \log \nu(n))$ bits.*

Proof: The following describes an algorithm which computes f assuming that there is a subroutine that solves solitude detection.

1. Each processor starts by sending its own input bit and forwarding $2T - 2$ more bits to its successor.

2. Each processor determines whether its sequence of $2T$ known bits is consistent with the configuration being well-formed. If this is so, it is *locally well-formed.* Each processor whose $2T$ known bits have the form $(1 \cdot 1 \cdot (0 \cdot \{0,1\}^t)^2$, determines if the configuration is *locally sequential*, that is, whether the consecutive blocks encode successive integers mod $\nu(n)$.

3. A processor is *distinguished* if either 1) it has determined that the configuration is not locally well-formed or 2) it has determined that the configuration is not locally sequential.

4. Distinguished processors initiate the solitude detection algorithm.

5. Upon termination of solitude detection, distinguished processors forward "function value is 1" to the next distinguished processor if solitude is confirmed and "function value is 0" otherwise.

It is easy to see that a configuration is well-formed if and only if it is everywhere locally well-formed. If a configuration is not well formed, it must be not locally well-formed in

more than one place. Therefore, there is one distinguished processor if and only if the configuration is almost sequential, and there is always at least one distinguished processor.

The first step requires the transmission of $n(2T-1) = O(n \log \nu(n))$ bits. The last step requires $O(n)$ bits. Since the only other communication is due to the solitude detection algorithm, the reduction requires $O(n \log \nu(n))$ bits. ■

Corollary 5.10 *The nonconstant cyclic boolean function f can be computed by a Las Vegas algorithm in expected complexity $O(n \log \nu(n)) + O(n\sqrt{\log n}) = O(n\sqrt{\log n})$ bits on a ring of size n.*

5.4 Extensions to Monte Carlo Function Evaluation?

It is natural to ask if the two results presented in this chapter have analogues in models that permit error with probability at most $1-\epsilon$. Only partial success in extending the nondeterministic lower bound to the nondeterministic/probabilistic model has been achieved. There is a natural extension of the function with an efficient Las Vegas solution to one with an efficient Monte Carlo solution. For completeness, these results are reported here. The relationships between these extensions and their counterparts in this chapter are briefly described; however the extensions are quoted without proof.

Algorithm *SD* in section 5.3 can be generalized to a Monte Carlo algorithm that solves solitude detection with confidence at least $(1-\epsilon)$ and has complexity $O(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}, \log\log(1/\epsilon)))$ expected bits on rings of known size n (see [4]). The function described in section 5.3 can be computed probabilistically using the same reduction as presented in that section followed by a Monte Carlo version of solitude detection. The complexity of evaluating this function with confidence $1-\epsilon$ is at most the complexity of the reduction plus the complexity of Monte Carlo solitude detection. Since the reduction requires $O(n \log \nu(n))$ bits, this total is $O(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}))$. As will be seen in chapter 7, AND can be computed with confidence $1-\epsilon$ in $O(n \log\log(1/\epsilon))$. Combining these upper bounds yields:

Theorem 5.11 *There is a nonconstant cyclic boolean function that can be computed by a Monte Carlo algorithm in expected complexity* $O(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}, \log\log(1/\epsilon)))$ *bits on a ring of size n.*

A more elaborate version of theorem 5.7 provides a lower bound for function evaluation with confidence $1-\epsilon$ of $\Omega(n \min(\sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}))$ expected bits. This proof applies in the nondeterministic/probabilistic model and, as previously indicated, requires a much more intricate set of tools than those used here for the purely nondeterministic model. See [4] for a description of these tools. The same paper, [4], uses these tools to prove a lower bound of $\Omega(n \min(\log \nu(n) + \sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}, \log\log(1/\epsilon)))$ expected bits for solitude detection on rings of known size n — a bound that is achieved by the Monte Carlo algorithm for solitude detection. The proof of the solitude detection lower bound is achieved by combining two separate lower bounds, one of which is $\Omega(n \min(\sqrt{\log\log(1/\epsilon)}, \sqrt{\log n}))$ expected bits. The proof of the claimed function evaluation lower bound uses the same approach as the proof of the solitude detection lower bound in [4].

The upper and lower bounds match to within a constant factor only if $\log \nu(n) = O(\sqrt{\log\log(1/\epsilon)})$. The complexity of Monte Carlo function evaluation remains an open question when this condition is not met.

Chapter 6

Evaluation of Specific Functions I: Unknown Ring Size

The $\Omega(n\sqrt{\log n})$ general lower bound for Las Vegas evaluation of nontrivial functions was shown, in chapter 5, to be best possible by presenting a function that can be computed in $O(n\sqrt{\log n})$ expected bits. Similarly, in [21], a nontrivial function is constructed that can be evaluated deterministically in $O(n \log n)$ bits. However, both these examples of low complexity functions are somewhat contrived. The inherent complexity of natural boolean functions such as AND is not adequately addressed by these results.

Though functions are usually thought to have a fixed number of arguments, some common ones such as AND, OR, SUM and PARITY are easily generalized to have an arbitrary number of inputs. This chapter examines the inherent complexity of algorithms that evaluate these common functions over a range of ring sizes. The case of fixed ring size is the subject of chapter 7. For concreteness, the results are presented for algorithms that compute AND.

$$\mathrm{AND}(i_1, \ldots, i_n) = \begin{cases} 1 & \text{if } i_1 = i_2 = \cdots i_n = 1 \\ 0 & \text{otherwise} \end{cases}$$

The results, with some exceptions and changes as noted in section 6.3, are easily seen to generalize to include the functions listed above.

6.1 Upper Bounds

Common functions can be easily computed by a deterministic algorithm in $O(n^2)$ bits, on rings of size n, and this is known to be optimal [10]. For AND and OR, these results generalize to $\Theta(N^2)$ where N is an upper bound on ring size. For PARITY and SUM, no generalization is possible even to intervals of size $[N-1, N]$.

A leader, however, can be elected on a ring of size n, where $N \leq n \leq 2N-1$, in $O(n \log n)$ expected bits by a Las Vegas algorithm (chapter 2). As is described in chapter 3, any function that recognizes a regular set can be reduced to leader election in $O(n)$ bits. Therefore, AND, OR, and PARITY can be evaluated in $O(n \log n)$ expected bits on any ring of size n where n is known to within a factor of two, using randomization. The same technique can be used to compute SUM, however the reduction may require $O(n \log S)$ bits, where S is the value of the SUM.

Even with no knowledge of the ring size, n, the following deterministic algorithm is easily seen to evaluate AND in $O(n)$ bits.

```
Algorithm AND1:
   value ←input;
   IF value = 0 THEN
      send(< 0 >); stop.
   ELSE
      receive(< msg >);
      value ←0; send(< 0 >); stop.
```

Notice that when all message traffic has ceased, each processor has the correct value for AND in its local variable *value*. When the input is 1^n, however, processors cannot detect the end of the computation. Hence, this algorithm achieves only nondistributive termination.

As mentioned, if only an upper bound, N, is known for the ring size, n, AND can be evaluated with distributive termination by a deterministic algorithm. Each processor sends its input value and messages are forwarded for N rounds. By this point, each processor has seen all the input values on the ring and so can compute the AND of these

values. By generalizing the straightforward $\Omega(n^2)$ messages lower bound for AND on rings of known size, n, in [10], it is easy to see that any deterministic algorithm for AND requires $\Omega(N^2)$ messages when N is an upper bound on ring size. Using attrition, a Las Vegas algorithm for AND can be constructed that improves upon this deterministic result. Recall that simple attrition (chapter 2) sets contenders to noncontenders by the exchange of messages containing random coin tosses. The following algorithm is described for contenders. Noncontenders participate by incrementing the count message received in step 3 and forwarding it, by combining their input value with the *and* message when they receive it, and by simply forwarding other messages.

`Algorithm` *AND2*(N):

1. Initially all processors are contenders.

2. Contenders run simple attrition for $3 \log N$ rounds. Let $\pi_1, \ldots, \pi_m$ denote the remaining contenders.

3. Each contender π_i initiates a gap counter g_i which is incremented and forwarded until it is received by the next contender.

4. Each contender forwards the value of its gap counter to its contending successor. Contender π_i has the values of the two gaps, g_{i-1} and g_{i-2}, that precede it.

5. If $g_{i-1} = g_{i-2}$ then π_i sends a constant length *and*-message initialized to π_i's input value. The *and*-message accumulates the AND of the processors' input values as it is circulated around the ring. If $g_{i-1} \neq g_{i-2}$ then π_i initiates an *alarm*-message, waits to receive any message, and returns to step 2.

6. Each contender alternately receives and sends *and*-messages until either:

   (a) An *alarm* is received, or

   (b) $\left\lceil \frac{N}{g_{i-1}} \right\rceil$ *and*-messages have been received.

7. If case 6b occurs, then the function value $f(I)$ is set equal to the AND of all *and*-messages received. Forward $f(I)$ and stop. If case 6a occurs, then the *alarm* is forwarded, the contender waits to receive any message and returns to step 2.

Theorem 6.1 *Algorithm AND2 evaluates AND on all rings of size $n \leq N$ in expected bit complexity* $O(n \log N + N)$.

Proof:

Correctness: *AND2* terminates only when condition 6b is met, otherwise the processors return to step 2. If, after some pass through step 2, there is one remaining contender, say π_1, then the value of the counter g_1 after the subsequent execution of step 3, is equal to the ring size. In step 4, this counter is forwarded from π_1 to itself, and no *alarms* are generated. Hence, the condition of 6b will be met after $\lceil N/n \rceil$ more rounds. As demonstrated in chapter 2, with probability 1 attrition will reduce the number of contenders to one. Thus, if *AND2* does not terminate early when there are two or more remaining contenders, the attrition of step 2 guarantees that termination will occur with probability 1.

Suppose that at step 6, π_i receives $k = \left\lceil \frac{N}{g_{i-1}} \right\rceil$ *and*-messages. Then there were no alarms in the previous k messages and hence the previous k gaps must all be equal to g_{i-1}. Since no more than $\left\lfloor \frac{N}{g_{i-1}} \right\rfloor$ gaps of size g_{i-1} can fit on the ring, π_i has collected the *and*-messages from the entire ring. Therefore, upon termination, the contenders have correctly computed the AND of the input values. This value is forwarded, ensuring correctness for all processors.

Complexity: Let random variable Y_n be the bit complexity of *AND2* on a ring of size n. Let a *pass* of *AND2* be one execution of steps 2 through 6, and let random variable Z_n be the number of passes on a ring of size n. Define random variable $X_{i,n}$ to be the number of contenders remaining after i rounds of attrition in the execution of *AND2*, on a ring of size n if at least i rounds occur in the execution of *AND2*, and define $X_{i,n}$ to be 1 otherwise. Because there are n contenders at the beginning of the algorithm, $X_{0,n} = n$.

By the analysis of attrition in section 2.2,

$$\mathrm{E}(X_{i+1,n}) = (3/4)\mathrm{E}(X_{i,n}) + (1/4)\Pr(X_{i,n} = 1).$$

Thus, for rings of size $n > 2$:

$$\begin{aligned}
\mathrm{E}(X_{1,n}) &= \frac{3}{4}n \\
\mathrm{E}(X_{2,n}) &= \left(\frac{3}{4}\right)^2 n + \frac{1}{4}\Pr(X_{1,n} = 1) \\
\mathrm{E}(X_{3,n}) &= \left(\frac{3}{4}\right)^3 n + \frac{1}{4}\left[\Pr(X_{2,n} = 1) + \frac{3}{4}\Pr(X_{1,n} = 1)\right] \\
&\vdots \\
\mathrm{E}(X_{r,n}) &= \left(\frac{3}{4}\right)^r n + \frac{1}{4}\left[\Pr(X_{r-1,n} = 1) + \frac{3}{4}\Pr(X_{r-2,n} = 1)\right. \\
&\qquad \left. +, \ldots, + \left(\frac{3}{4}\right)^{r-2}\Pr(X_{1,n} = 1)\right] \\
&< \left(\frac{3}{4}\right)^r n + \frac{1}{4}\sum_{i=0}^{\infty}\left(\frac{3}{4}\right)^i < \left(\frac{3}{4}\right)^r n + 1
\end{aligned}$$

Since $n \leq N$, $\mathrm{E}(X_{3\log N,n}) \leq 1 + N^{-0.245}$. For any random variable W, $\mathrm{E}(W) > \Pr(W = 1) + 2(1 - \Pr(W = 1))$. Therefore, $\Pr(X_{3\log N} = 1) > 1 - N^{-0.245}$. From this it follows that $\mathrm{E}(Z_n) < \frac{1}{1-N^{-0.245}}$ if $n \geq 2$.

Suppose the first pass has t remaining contenders at the end of step 2. Then it is easily confirmed that the complexity of steps 2 through 5 is $\mathrm{O}(n \log N)$ bits, while that of step 6 is $\mathrm{O}(n\left\lceil\frac{N}{n/t}\right\rceil) = \mathrm{O}(tN)$. Since subsequent passes have at most t contenders, the complexity of m passes with t contenders after the first pass, is $\mathrm{O}(m(n \log N + tN))$ bits. Suppose this is at most $k \cdot m(n \log N + tN)$. (The constant k depends upon the encoding used in *AND2* but $k = 10$ is easily seen to be sufficient.) Then:

$$\begin{aligned}
\mathrm{E}(Y_n) &= \sum_{m=1}^{\infty}\mathrm{E}(Y_n|Z_n = m)\Pr(Z_n = m) \\
&= \sum_{m=1}^{\infty}\left[\sum_{t=1}^{n}\mathrm{E}(Y_n|Z_n = m \wedge X_{3\log N,n} = t)\Pr(X_{3\log N,n} = t)\right]\Pr(Z_n = m) \\
&\leq \sum_{m=1}^{\infty}\left[\sum_{t=1}^{n} km(n \log N + tN)\Pr(X_{3\log N,n} = t)\right]\Pr(Z_n = m)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{m=1}^{\infty}\left[\sum_{t=1}^{n}(kmn\log N)\Pr(X_{3\log N,n}=t)\right]\Pr(Z_n=m)\\
&\quad+\sum_{m=1}^{\infty}\left[\sum_{t=1}^{n}(kmtN)\Pr(X_{3\log N,n}=t)\right]\Pr(Z_n=m)\\
&= \sum_{m=1}^{\infty}(kmn\log N)\Pr(Z_n=m)\\
&\quad+\sum_{m=1}^{\infty}kmN\left[\sum_{t=1}^{n}t\Pr(X_{3\log N,n}=t)\right]\Pr(Z_n=m)\\
&= (kn\log N)\mathrm{E}(Z_n)+kN\sum_{m=1}^{\infty}m\mathrm{E}(X_{3\log N,n})\Pr(Z_n=m)\\
&= k\mathrm{E}(Z_n)\left[n\log N+N\cdot\mathrm{E}(X_{3n\log N,n})\right]\\
&= O(n\log N+N)
\end{aligned}
$$

■

Note that AND (and similarly OR) remain computable in a situation where leader election is impossible. Such a general algorithm for rings of size $n \leq N$ does not exist for PARITY (and hence not for SUM) as will be seen in section 6.3.

6.2 Lower Bounds for AND

The previous section presented only a nondistributively terminating algorithm for AND when there is no upper bound on ring size. The next theorem shows that this is the best that can be expected. It is convenient for the lower bound results to interpret a boolean function f as the characteristic function of a language $L \subseteq \bigcup_{n\in[a,b]}\{0,1\}^n$. Algorithms that compute f are algorithms that recognize L.

Theorem 6.2 *If α is a distributively terminating nondeterministic algorithm that computes AND for arbitrarily large rings, then no computation of α strongly evaluates AND on input 1^n, for any n.*

Proof: Suppose, for some n, that $C = h_1, \cdots, h_n$ is the computation of $\pi = \pi_1, \ldots, \pi_n$ where $\pi_i \in \alpha(1)$, $n > N$, and π strongly evaluates AND. Since the input is 1^n, each h_i

must be an accepting history. Let $|h_1| = k$. Consider the computation $g_1, \ldots, g_m$ of the ring $\rho = (\pi)^{k+1}\tau$ where $\tau \in \alpha(0)$. By lemma 5.4, the $(k+1)^{th}$ copy of π_1 in ρ realizes a history, g_{kn+1}, such that the first k messages in g_{kn+1} are the same as the first k messages in h_1. Since the computation terminates distributively, this copy of π_1 accepts the input. This is a contradiction since the input of the ring contains a 0. ■

The rest of this section assumes that there is a bound, N, on ring size, since this is necessary to achieve distributive termination. The goal is to show that there is no Las Vegas algorithm for AND whose complexity, even in the best case, improves upon the expected complexity of algorithm *AND2*.

In order to compute AND with distributive termination, the trace of at least one message envelope must be at least equal to the ring size. Hence, there is a trivial lower bound of $\Omega(n)$ messages for AND. This can be strengthed to $\Omega(N)$ messages, as is shown in the next theorem.

Theorem 6.3 *Every nondeterministic algorithm that computes AND for all rings of size $n \leq N$ has complexity $\Omega(N)$ messages for all $n \leq N$.*

Proof: If $n \geq N/2$, then the $\Omega(n)$ lower bound suffices so assume $n < N/2$. Let α be a nondeterministic algorithm that computes AND and suppose that for some $n < N/2$, α has message complexity less than $N/2$. Then there is some computation, C, such that $C = h_1, \cdots, h_n$ is the computation of $\pi = \pi_1, \ldots, \pi_n$ where $\pi_i \in \alpha(1)$ and $\pi_1, \ldots, \pi_n$ strongly evaluates AND on input 1^n and $|C| < N/2$. Then for some i, $|h_i| < N/(2n)$. Let $k = \left\lfloor \frac{N-1}{n} \right\rfloor$. Then the computation of the sequence $\rho = (\pi)^k\gamma$, where $\gamma \in \alpha(0)$, is a possible computation of α on a ring of size $m = nk + 1 \leq N$ with input $1^{nk}0$. But $k \geq N/(2n)$. So, by lemma 5.4, the last copy of π in ρ has histories $h'_1, \ldots, h'_n$, each of which match the histories in C for at least the first $N/(2n)$ messages. Since the computation terminates distributively, the k^{th} copy of π_i accepts the input after these $N/(2n)$ messages. Since the input contains a 0, this contradicts the assumption that α computes AND on rings of size $m = nk + 1 \leq N$. ■

Attrition is at the heart of both the leader election algorithm in section 2.4 and the AND algorithm in section 6.1. Algorithms for other common problems can also be constructed from attrition. The pervasiveness of potential applications of attrition is perhaps not surprising. Both feasibility and efficiency are affected by randomized attrition. For some problems, a solution is not possible while symmetric configurations persist. Symmetry can be broken by attrition. Communication (measured either as messages or bits) is reduced by ensuring that processors do not duplicate the work of others. Attrition furthers this end by avoiding redundant messages.

Recall that an attrition procedure, with probability 1, reduces the number of contenders to one without any possibility of eliminating all contenders. It is frequently useful to have such a procedure when constructing algorithms. A closely related problem, *envelope-attrition*, is the problem of eliminating all but one message envelope. Note that the algorithm for attrition that is introduced in chapter 2 and used in subsequent chapters, solves envelope-attrition as well as (contender) attrition. For the purpose of proving lower bounds, the envelope-attrition problem is more convenient. Unless stated otherwise, envelope-attrition is assumed to be the problem of reducing the number of envelopes from n to one, where n is the ring size. That is, all processes are assumed to be initiators.

As will be seen, attrition is not just useful but is, in fact, an essential part of AND in the sense that an efficient algorithm for AND implies an efficient algorithm for envelope-attrition. Lower bounds for both the best case and the expected case of AND can be derived from the corresponding lower bounds for envelope-attrition.

The envelope-attrition lower bound that follows, holds for a less general model than the nondeterministic model of the previous theorem. It will be shown in chapter 7 that both envelope and contender attrition can be done very efficiently if deadlock is permitted even with low probability. But nondeterminism permits deadlock. Therefore, under the full power of nondeterminism, AND would inherit only a weak lower bound from envelope-attrition. By forbidding deadlock, the desired lower bound can be achieved. The nondeadlocking restriction imposed upon nondeterministic algorithms, results in a model that characterizes the best case execution, over all possible random experiments, of a Las

Vegas algorithm.

Recall that the complexity of an algorithm is defined as the worst case over all possible inputs. Hence, the *best case complexity* of a Las Vegas message-driven algorithm, α, is the maximum over all inputs, I, of the minimum over all computations, C of α on I, of the complexity of C.

Lemma 6.4 *Let α be a Las Vegas algorithm for AND on rings of size $n \in [a,b]$ with complexity $f(n)$ bits in the best case. Then there is an envelope-attrition procedure for rings of size $n \in [a,b]$ that has best case complexity at most $O(n) + f(n)$ bits.*

Proof: Let γ denote the simple attrition procedure of chapter 2. Then γ is both an envelope and a contender attrition procedure.

```
Procedure ATTRITION1:
1. choose a bit, mybit ∈ {0,1};
2. IF mybit = 0 THEN
      create an envelope and initiate γ;
      henceforth participate in γ and discard all α messages
   ELSE
      initiate α;
      Participate in α only as long as no γ message arrives;
      Upon receipt of a γ message, participate in γ as a
      noncontender and discard all subsequent α messages;
      IF α confirms ''all 1's'' THEN initiate γ.
```

Step 2 performs envelope-attrition on the set of processors that choose 0 in step 1. In the event that there were no attrition participants, all processors run α. On input 1^n α confirms "all 1's" and thus ensures that all processors initiate simple attrition. Thus, *ATTRITION1* is an envelope-attrition procedure.

The number of bits sent by *ATTRITION1* in the best case is no more than the number of bits communicated when exactly one processor generates 0 in step 1. In this situation, the best case complexity of *ATTRITION1* is $O(n)$ bits for attrition plus at most $f(n)$ bits for the best case execution of α on input $1^{n-1}0$. ∎

This reduction is similar to a more general reduction in chapter 7. However, this one is simpler because there is no requirement to accommodate algorithms that permit error and because the cost attributed to the reduction is the cost assuming a favourable outcome of the arbitrary choices in step 1. The object is to prove a lower bound on the best case complexity of any Las Vegas algorithm for AND by proving the required lower bound for the best case of envelope-attrition. But best case lower bounds carry across best case reductions. Thus, it suffices to analyse the cost of the reduction in the best case. A superlinear lower bound for the best case of an envelope-attrition computation implies the same lower bound for the best case of a Las Vegas AND computation.

It only remains to bound the best case bit complexity of envelope-attrition.

A lemma is isolated from the proof because it is useful again in chapter 7. The definition of a computation can be extended to include *partial computations* by allowing a scheduler to suspend the sending of messages temporarily before all message traffic has ceased. In a suspended computation, message envelopes that are not yet terminated are assumed to reside at processes rather than on links. Though a ring of deterministic processes has only one (complete) computation, there are many possible partial computations.

Lemma 6.5 *Let* $C = h_1, \cdots, h_n$ *be any partial computation of* $\pi = \pi_1, \ldots, \pi_n$ *where* $|h_i| = |h_j|$ *for some* $i < j$. *Refer to* $\rho_1 = \pi_1, \ldots, \pi_i, \pi_{j+1}, \ldots, \pi_n$ *and* $\rho_2 = \pi_{i+1}, \ldots, \pi_j$ *as* segments. *Then for each segment* ρ_i, $i = 1, 2$, *the number of envelopes initiated by* ρ_i *is equal to the number of envelopes terminated by* ρ_i *plus the number of envelopes suspended in* ρ_i.

Proof: Any message envelope that is both initiated and terminated or suspended by the same segment contributes an equal number of messages to h_i and h_j. Each message envelope that is initiated by ρ_1 and terminated or suspended by ρ_2 contributes one more message to h_i than to h_j. Similarly, each message envelope that is initiated by ρ_2 and terminated or suspended by ρ_1 contributes one more message to h_j than to h_i. Since h_i and h_j have an equal number of messages, there must be the same number of message envelopes of each kind. ∎

Theorem 6.6 *Let α be any envelope-attrition procedure for rings of size $n \in [(1-\frac{1}{k})N, N]$ where $k \geq 1$. Then, for all rings of size $n \in [(1-\frac{1}{k})N, N]$, the best case bit complexity of α is $\Omega\left(\frac{n \log n}{k}\right)$.*

Proof: Let $n \in [(1-\frac{1}{k})N, N]$. The mapping from inputs to sets of processes that defines α degenerates to just one set of processes because there are no inputs for envelope-attrition and all processors are initiators. Let $\pi = \pi_1, \ldots, \pi_n$ be a sequence of deterministic processes in α. Suspend the computation of π when it first occurs that exactly one message envelope remains and the envelope has returned to its initiator, π_e. Let $C = h_1, \cdots, h_n$ be the partial computation of $\pi = \pi_1, \ldots, \pi_n$ up to this point. Suppose that $\|C\| < (n \log n)/(3k)$. Since at least $(n \log n)/2$ bits are required to encode n distinct histories, there exists $i \neq j$ such that $h_i = h_j$. If $e \notin [i+1, j]$, then let $\pi' = \pi_{i+1}, \ldots, \pi_j$, otherwise let $\pi' = \pi_{j+1}, \ldots, \pi_n, \pi_1, \ldots, \pi_i$. Ring π' produces a computation such that each process $\pi'_l \in \pi'$ realizes a history h'_l that is a prefix of its history in π. The suspended message envelope is not in segment π'. It follows, from lemma 6.5, that the number of message envelopes initiated by segment π' equals the number terminated by π'. Therefore, the computation of π' terminates all message envelops.

Further shrink π' by repeatedly removing segments between identical histories until all remaining histories are distinct. The resulting sequence, γ, also terminates all message envelopes. Denote the length of γ by a. By lemma 5.1, $a \leq \frac{n \log n}{k \log(n \log n/(3k))} < \frac{n}{k}$ for large enough N.

By replicating γ $\lfloor N/a \rfloor$ times, a new sequence of processes, δ, is obtained with length within $\left[\left(1-\frac{1}{k}\right)N, N\right]$. But the computation of δ is a possible computation of α. This computation deadlocks eliminating all message envelopes, contradicting the correctness of α. ∎

Corollary 6.7 *Every Las Vegas algorithm that computes AND for all rings of size $N/2 \leq n \leq N$ has complexity $\Omega(n \log n)$ bits even in the best case for any $n \in [N/2, N]$.*

Notice that the upper and lower bounds for AND match. If $n > N/\log N$ then the lower bound is $\Omega(n \log n)$ bits. But this is $\Omega(n \log N)$ for these values of n. If $n \leq N/\log N$, then the lower bound of $\Omega(N)$ messages dominates. This is achieved by algorithm *AND2* which has complexity $O(N)$ expected bits for these values of n.

6.3 Extensions to PARITY

Section 6.1 describes how a simple Las Vegas algorithm for PARITY is constructed from leader election. Leader election is feasible as long as ring size is known to within a factor of two. Thus, knowledge of ring size to within a factor of two is sufficient to compute PARITY. The next theorem show that the same knowledge of ring size is necessary for any, even nondistributively terminating, algorithm that computes PARITY.

Theorem 6.8 *Let α be any (even nondistributively terminating) nondeterministic algorithm that computes PARITY for rings of size $n \in [N/2, N]$. Then for every odd parity input, I, on rings of size $N/2$, there is no computation of α that strongly evaluates PARITY on input I.*

Proof: Let $I = i_1, \ldots, i_{N/2}$ be any input with odd parity. Let $C = h_1, \ldots, h_{N/2}$ be the computation of $\pi = \pi_1, \ldots, \pi_{N/2}$ where $\pi_j \in \alpha(i_j)$ and π strongly evaluates PARITY on input I. Each h_i must be an accepting history. By lemma 5.3, the computation of the sequence $\pi' = \pi\pi$ is $C' = h_1, \ldots, h_{N/2}, h_1, \ldots, h_{N/2}$ where each history is accepting. This is a possible computation of α on a ring of size N with input II which has even parity, contradicting the assumption that α computes PARITY on rings of size N. ■

Since algorithms for SUM can be used to compute PARITY, the same amount of knowledge of ring size is required to compute SUM.

Lemma 6.4 relates the complexity of AND to the complexity of envelope-attrition. Similar reductions relate the complexity of many other functions to that of envelope-attrition. Let f be any function that has the property that for all inputs some message must have

trace at least n. An efficient procedure for envelope-attrition can be constructed from an efficient algorithm for f in a manner similar to procedure *ATTRITION1*. The algorithm for f replaces the algorithm for AND in that reduction. If f terminates, then any envelope-attrition algorithm is initiated. It is easily checked that the procedure is an envelope-attrition procedure and that the cost of the reduction in the best case remains $O(n)$ bits. Therefore, the $\Omega(n \log n)$ best case lower bound for envelope-attrition extends to best case lower bounds for PARITY for rings of size $n \in [N, 2N-1]$. Clearly, SUM and OR inherit the same lower bound from envelope-attrition.

6.4 Summary

The best case nature of the $\Omega(n \log n)$ lower bound for AND, OR, PARITY, and SUM leaves some room for improvement. These lower bounds are based on the same lower bound for deadlock free envelope-attrition. Algorithms for rings of size $n \in [N/2, N]$ appear to require $\Omega(n \log n)$ bits in the best case to strongly evaluate AND on input 1^n even if deadlock is permitted. Although envelope-attrition can be efficient if deadlock is allowed even with small probability, it seems that $\Omega(n \log n)$ bits are needed just to verify that a function value of 1 is correct. Nondeterministic algorithms are permitted to deadlock. Therefore, what is needed is an extension of the existing lower bound to the more powerful nondeterministic model; but this extension appears to require new techniques.

The following table provides a summary of the improvements over deterministic algorithms that are provided by Las Vegas algorithms for computing functions such as AND and PARITY. It also underscores the fact that Las Vegas algorithms for these functions require essentially the same amount of communication in the best case as on average. The nondeterministic $\Omega(N)$ messages lower bound for AND is achieved even in the average case using constant length messages. Thus, this term contributes only $O(N)$ expected bits of communication to the upper bound.

	deterministic	Las Vegas — expected case	Las Vegas — best case
AND and OR on rings of size $n \leq N$	$\Theta(N^2)$ bits	$O(n \log n + N)$ bits	$\Omega(n \log n)$ bits + $\Omega(N)$ messages
PARITY on rings of size $N/2 \leq n < N$	impossible	$O(n \log n)$ bits	$\Omega(n \log n)$ bits

Chapter 7

Evaluation of Specific Functions II: Known Ring Size

It was shown in chapter 6 that any algorithm for AND has complexity $\Theta(n \log n)$ expected bits when it must work for $n \in [N/2, N]$. The results of chapter 5 rely on a problem, solitude detection, whose complexity, given exact knowledge of ring size, drops from $\Theta(n \log n)$ to $\Theta(n\sqrt{\log n})$ expected bits. The question arises whether number theoretic properties of the ring size might also be used to reduce the complexity of functions like AND when the ring size is fixed.

This chapter shows:

1. The best case complexity of AND drops from $\Theta(n \log n)$ bits when ring size n is known to within a factor of two, to $\Theta(n\sqrt{\log n})$ bits when n is known exactly.
2. The expected complexity of AND remains $\Theta(n \log n)$ messages when ring size n is known exactly.

There is a nondeadlocking attrition procedure for rings of fixed size n, that has best case complexity $O(n \log \log n)$ bits. This attrition procedure can be combined with the efficient solitude detection algorithm of section 5.3 to yield a leader election algorithm (and hence

an AND algorithm) for rings of fixed size, with best case complexity $O(n\sqrt{\log n})$ bits. The analysis of the best case execution of attrition and the implications of the result are contained in section 7.1.

The $\Omega(n \log n)$ lower bound for AND on rings of size $n \in [N/2, N]$ was achieved by supplying the same lower bound for the best case execution of nondeadlocking envelope-attrition together with a reduction from envelope-attrition to AND. Because of the existence of an attrition procedure for ring of known size with low best case complexity, this approach cannot be extended to AND on rings of known size. Furthermore, all the lower bound techniques employed in previous chapters apply to the best case of Las Vegas algorithms. Therefore, they cannot possibly provide a lower bound of $\Omega(n \log n)$ expected bits for Las Vegas AND on fixed size rings. New tools capable of distinguishing Las Vegas algorithms from nondeterministic ones are needed to achieve this goal. Such tools can be constructed from the techniques recently introduced by Duris and Galil in [14]. They prove that:

> The average number of messages required by any *deterministic* algorithm that elects a leader on an asynchronous bidirectional ring with distinct identifiers, where ring size n is known and is a power of 2, is $\Omega(n \log n)$, for any sufficiently large identifier set.

This result implies the same lower bound for the expected message complexity of any Las Vegas leader election algorithm for an anonymous ring of known size $n = 2^k$, as is now demonstrated. Let α be a Las Vegas leader election algorithm for a ring of size $n = 2^k$. Suppose that with very high probability, no processor uses more than $f(n)$ random bits when running α. Now consider the class $\mathcal{R}$ of rings of size n with distinct identifiers taken from the interval $[1, 2^{f(n)}]$. A deterministic algorithm β for $\mathcal{R}$ can be constructed from α by using the bits of the processor identifiers in place of the random bits. In the rare event that some processor requires more pseudo-random bits than provided by the identifiers, β proceeds by running any $O(n \log n)$ messages deterministic leader election algorithm. Algorithm α must have complexity $\Omega(n \log n)$ expected messages since otherwise algorithm

β would contradict the $\Omega(n \log n)$ messages lower bound for deterministic leader election.

Alternatively, an $\Omega(n \log n)$ expected message lower bound for nondeadlocking envelope-attrition can be achieved directly using modifications of the techniques of Duris and Galil. In the deterministic model, counting arguments are used to ensure that the characteristic of distinct identifiers is maintained. In the randomized model, these combinatorial techniques are not needed. As a consequence, the proof of a lower bound of $\Omega(n \log n)$ expected messages for nondeadlocking envelope-attrition when n is a power of two, is significantly simpler than the corresponding proof of a lower bound of $\Omega(n \log n)$ messages on average for deterministic leader election. The simplicity facilitates extension of the result in two directions; namely, for fixed ring sizes that are not a power of two, and for algorithms that permit error with probability at most ϵ. An envelope-attrition procedure that deadlocks with probability at most ϵ is called ϵ-attrition. Specifically, a procedure for *ϵ-attrition* must, with probability $1 - \epsilon$, eliminate all but one message envelope. It is shown, in section 7.2, that every ϵ-attrition procedure for rings of known size n has expected complexity $\Omega\left(n \min\left\{\log n, \log\log\left(1/\epsilon\right)\right\}\right)$ messages.

A Monte Carlo AND algorithm that errs with probability at most ϵ can be converted into a ϵ-attrition procedure. Section 7.3 contains a reduction from ϵ-attrition to Monte Carlo AND that permits the ϵ-attrition lower bound to extend to AND and numerous other functions. The section also describes simple Monte Carlo algorithms for these problems, which demonstrate that the lower bound is tight. A large number of problems including AND, OR, PARITY, SUM and leader election are thus shown to have expected complexity $\Theta\left(n \min\left\{\log n, \log\log\left(1/\epsilon\right)\right\}\right)$ messages on rings of known size n.

7.1 Best Case Attrition

Recall that by definition a nondeadlocking attrition procedure runs forever, and hence that the communication complexity of an attrition computation is defined as the number of bits exchanged until one message envelope remains. Let $\nu(n)$ be the smallest positive integer that does not divide n. Note that $\nu(n) = O(\log n)$ (see [22]).

Theorem 7.1 *There is a nondeadlocking attrition procedure for rings of known size, n, that has best case complexity $O(n \log \nu(n))$ bits.*

Proof: The following describes such as procedure. Initially all processors are contenders. Each processor in the ring randomly chooses a number in $\{0, 1, \ldots, \nu(n)-1\}$ and sends that number to its successor. If a processor sends l and receives $(l+1) \bmod \nu(n)$, it becomes a non-contender and sends no message. The algorithm proceeds by running simple attrition on the remaining contenders after this first round. Since $\nu(n)$ does not divide n, it is not possible for all processors to become non-contenders in the first round. But there is a computation of this procedure (for example, processor π_i chooses $i \bmod \nu(n)$) that leaves exactly one contender and one message envelope after the first exchange of messages. The bit complexity of this computation is $O(n \log \nu(n))$. ∎

As was the case for other attrition procedures, the attrition procedure of theorem 7.1 solves both envelope and contender attrition. The following theorem demonstrates that this is the best that an envelope-attrition procedure could achieve.

Theorem 7.2 *Every nondeadlocking envelope-attrition procedure for rings of fixed size, n, has complexity $\Omega(n \log \nu(n))$ bits even in the best case.*

Proof: Let α be a nondeadlocking envelope-attrition procedure for rings of fixed size n. Let $\pi = \pi_1, \ldots, \pi_n$ be a sequence of deterministic processes in α. Suspend the computation of π when it first occurs that exactly one message envelope remains and it has returned to its initiator, π_e. Let $C = h_1, \cdots, h_n$ be the partial computation of $\pi = \pi_1, \ldots, \pi_n$ up to this point. Suppose that $\|C\| < (n \log \rho(n))/2$ where $\rho(n) = \nu(n) - 1$. Say that history h_i is *short* if $\|h_i\| < \log \rho(n) - 1$. Then at least half of the n histories are short. Starting at h_{e+1} partition π and its partial computation C into segments $\pi = \gamma_1, \ldots, \gamma_{n/\rho(n)}$ and $C = S_1, \ldots, S_{n/\rho(n)}$ where each γ_i has length $\rho(n)$ and each segment S_i contains $\rho(n)$ histories. Some segment S_k must contain at least $\rho(n)/2$ short histories. But there are fewer than $2^{\log \rho(n)-1} = \rho(n)/2$ distinct short histories. Hence, S_k must contain two identical short histories, say h_i and h_j. Let $a = j - i$. Because $a \leq \rho(n)$, a divides n. So the segment

$\pi_i, \ldots, \pi_{j-1}$ can be replicated n/a times to form a new ring δ of size exactly n, which is a possible sequence of processes in α. Since the suspended message envelope is not in δ, by lemma 6.5, the number of message envelopes initiated by δ equals the number terminated by δ. Therefore, the computation of δ terminates all messages. So a possible computation of α deadlocks, contradicting the assumption that α is an envelope-attrition procedure. ■

A Las Vegas leader election algorithm for rings of fixed size n which has best case complexity $O(n\sqrt{\log n})$ bits is formed by combining the attrition procedure of theorem 7.1 with algorithm *SD* of section 5.3. Call the first round of the attrition procedure described in theorem 7.1, $\nu(n)$-*attrition*. Specifically, this leader election algorithm is:

1. All processors run one round of $\nu(n)$-attrition.
2. Remaining contenders initiate algorithm *SD*.
3. If solitude is confirmed then the sole remaining contender is elected. If solitude is not confirmed then remaining contenders elect a leader using any leader election algorithm.

In the best case, the first step will reduce the contenders to one ($O(n \log \nu(n))$ bits) and this will be detected in step 2 ($O(n\sqrt{\log n})$ bits). Thus:

Theorem 7.3 *There is a Las Vegas leader election algorithm (and hence a Las Vegas AND algorithm) for rings of fixed size, n, with best case complexity* $O(n\sqrt{\log n})$ *bits.*

According to theorem 5.7, this is optimal even for nondeterministic algorithms.

7.2 Lower Bounds for ϵ-attrition

Up to this point, the discussion concerning attrition has focussed on nondeadlocking attrition — attrition procedures that never terminate all message envelopes. A more general notion of attrition (ϵ-attrition) permits deadlock with probably at most ϵ. This section bounds the expected message complexity of ϵ-attrition. It will be shown that any ϵ-attrition

procedure has expected message complexity $\Omega(n \min\{\log n, \log\log(1/\epsilon)\})$. The expected message complexity of nondeadlocking envelope-attrition ($\Omega(n \log n)$) is derived from the general result by setting the allowable probability of deadlock, ϵ, to less than $1/2^n$.

The lower bounds of chapter 5 and chapter 6 address bit complexity for either nondeterministic or best case Las Vegas computation. They apply to any (even non-message-driven) Las Vegas algorithm and hold for all possible schedulers. In contrast, what follows is a message complexity lower bound and applies only to the expected number of messages. Since the proof imposes constraints on the scheduler, the result holds for the worst case rather than for the best case over all schedulers, but also applies to even non-message-driven algorithms.

The proof relies on the model of a randomized algorithm as a mapping from input values to probability spaces of deterministic processes, a perspective developed in chapter 4. Because attrition has no inputs, a Monte Carlo envelope-attrition procedure (that is, an ϵ-attrition procedure) is interpreted as a single probability space of deterministic processes. The techniques used here are applicable to bidirectional rings, so the lower bound is presented in that generality. Although the detailed model of a deterministic process for a unidirectional ring, also in chapter 4, could be extended to a model for bidirectional rings, a coarser model suffices for the current result. This is because the unidirectional process model proved useful for examining bit complexity, but includes more detail than is necessary for proofs concerning message complexity. It is enough to assume that the behaviour of each process is entirely determined by its current state, where the current state records all information the process can know. Since this is all that is assumed of a process, the result applies even to processes that are not necessarily message-driven.

The proof uses two techniques that are adapted from those introduced by Duris and Galil [14]. The first technique argues that expected message complexity for parts of the ϵ-attrition procedure cannot be too low because otherwise deadlock will occur, under a specified scheduler, with intolerably high probability. This is the essence of lemma 7.4. The second technique, used here in theorem 7.5, sums these expected message complexities for disjoint parts of the ϵ-attrition procedure to get a lower bound on total expected message

complexity.

Definitions and Notation:

Let α be an ϵ-attrition procedure for anonymous rings of known size n. Let $\mathcal{P}$ be the probability space of deterministic processes associated with α. $\mathcal{P}^l$ denotes the product space formed from l copies of $\mathcal{P}$ together with the induced product probability measure. Let $\mathcal{R}$ be a ring of n processes each from $\mathcal{P}$ and let Π be any sequence of consecutive processes in $\mathcal{R}$. Let $len(\Pi)$ denote the number of processes in sequence Π. If $len(\Pi) = l$, then Π is called an *l-process*.

Imagine placing barriers on the links before and after sequence Π, and running α on Π with the barriers remaining in place until $\mathcal{R}$ is *quiescent*, that is, all remaining messages are queued at barriers and computation cannot proceed until at least one barrier is removed. (There is no loss of generality in assuming that with probability 1, a random sequence of processes reaches quiescence for every scheduler, because otherwise a large amount of message traffic can be forced.) A *computation of α on* Π is any pattern of message traffic on the segment Π that could occur from the beginning of the computation up to the point when $\mathcal{R}$ is quiescent, under any scheduler that respects the barriers before and after the sequence Π. A *partition* of Π is a sequence of subsequences of processes whose concatenation is Π. A partition of Π into subsequences $\Pi_1, \ldots, \Pi_k$ is denoted $\Pi_1|\Pi_2|\ldots|\Pi_k$. A *decomposition* for an integer L is any sequence $l_1, \ldots, l_k$ of positive integers that sum to L and is denoted $l_1|l_2|\ldots|l_k$. The partition $\Pi_1|\Pi_2|\ldots|\Pi_k$ is said to be *consistent* with decomposition $l_1|l_2|\ldots|l_k$ if $l_i = len(\Pi_i)$ for $1 \leq i \leq k$.

Suppose that barriers are placed on the links between adjacent members of some partition $\Pi_1|\Pi_2|\ldots|\Pi_k$ of Π. It is intended to measure the number of *additional* messages sent by α on sequence Π after removal of all the barriers between adjacent segments of the partition while the links at either end of Π remain blocked. The scheduler, however, is only partially constrained by the barriers. Several different computations may still arise depending on the scheduling of messages within each of the segments in the partition and on the

scheduling after the barriers are removed. A *scheduling function* is a function that assigns a fixed schedule to each sequence of processes. If $\mathcal{S}$ is a scheduling function, then $\mathcal{S}(\Pi)$ is a *scheduled process.* If Π is a sequence of deterministic processes, then there is exactly one computation that can arise from $\mathcal{S}(\Pi)$. The notation is extended so that $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$ denotes the schedule $\mathcal{S}$ applied separately to each process Π_i in $\Pi_1|\Pi_2|\ldots|\Pi_k$. To avoid the ambiguity caused by undetermined schedulers, the following will establish a scheduling function that assigns a fixed scheduled process $\mathcal{S}^*(\Pi)$ to a sequence Π.

Let $\mathcal{S}'$ be some scheduling function that schedules Π by first running $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$ and then removing the barriers between the segments of the partition $\Pi_1|\Pi_2|\ldots|\Pi_k$ and continuing with α until Π is quiescent. $\mathcal{S}'$ is an *extension of* $\mathcal{S}$ *for* $\Pi_1|\Pi_2|\ldots|\Pi_k$. The computation that occurs under $\mathcal{S}'$ after removal of the barriers in $\Pi_1|\Pi_2|\ldots|\Pi_k$ and up to the point where Π is quiescent, is a possible *continuation of* $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$. The continuation of $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$ that minimizes the number of messages sent during the continuation, is the *minimum continuation of* $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$. The *cost of the continuation of* $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$ is the number of messages sent during the minimum continuation of $\mathcal{S}(\Pi_1|\Pi_2|\ldots|\Pi_k)$.

As a first step toward defining $\mathcal{S}^*$, a fixed partition is specified for each sequence of processes. Let $L \leq n$ and let k be the integer satisfying $\left\lfloor \frac{n}{8^k} \right\rfloor \leq L < \left\lfloor \frac{n}{8^{k-1}} \right\rfloor$. Let $l = \left\lfloor \frac{n}{8^{k+1}} \right\rfloor$. Define the decomposition $D(L) = l_1|l_2|l_3|l_4|l_5|l_6$ by $l_1 = l_2 = l_3 - 1 = l_4 - 1 = l_5 = l$ and $l_6 = L - \sum_1^5 l_i$. The partition $\Pi_1|\ldots|\Pi_6$ of $\Pi \in \mathcal{P}^L$ that is consistent with $D(L)$ is denoted $\delta(\Pi)$.

Given a sequence $\Pi \in \mathcal{P}^L$, the decomposition, $D(L)$, and the corresponding partition, $\delta(\Pi)$, are used to associate a fixed scheduled process, $\mathcal{S}^*(\Pi)$, with Π. $\mathcal{S}^*(\Pi)$ is specified recursively by:

1. If $L < 8$ then $\mathcal{S}^*(\Pi)$ is the scheduler that minimizes the number of messages sent in a computation of α on Π.

2. Otherwise, $\mathcal{S}^*(\Pi)$ is the extension of $\mathcal{S}^*$ for $\delta(\Pi)$ that minimizes the number of messages sent in the continuation of $\mathcal{S}^*(\delta(\Pi))$.

The cost of the continuation of $\mathcal{S}^*(\Pi_1|\Pi_2|\ldots|\Pi_k)$ is denoted $CC(\Pi_1|\Pi_2|\ldots|\Pi_k)$.

Lemma 7.4 bounds the expectation of $CC(\Pi_1|\cdots|\Pi_6)$ when $\Pi_1|\cdots|\Pi_6$ is a partition of Π that is consistent with $D(L)$. That is, given that Π is a random L-process partioned consistently with $D(L)$ into $\Pi_1|\cdots|\Pi_6$, lemma 7.4 bounds the minimum, over all extensions of $\mathcal{S}^*(\Pi_1|\cdots|\Pi_6)$, of the expected number of messages sent in the continuation of the extension. In general, $CC(\Pi_1|\Pi_2|\ldots|\Pi_k)$ is dependent on the choice of Π as well as on the partition of Π. However, the expectation of $CC(\Pi_1|\Pi_2|\ldots|\Pi_k)$ depends only on the decomposition $len(\Pi_1)|\ldots|len(\Pi_k)$. To emphasize this dependence, define $\mathrm{E}_{cc}(l_1|\ldots|l_k)$ to be the expected value of $CC(\Pi_1|\Pi_2|\ldots|\Pi_k)$ over all processes Π such that $\Pi_1|\Pi_2|\ldots|\Pi_k$ is a partition of Π consistent with $l_1|\ldots|l_k$.

If $S \subseteq \mathcal{P}^l$, then $\Pr(S)$ is used to abbreviate $\Pr(x \in S | x \in \mathcal{P}^l)$.

Lemma 7.4 *For any ϵ-attrition procedure for anonymous bidirectional rings of fixed size n and for all $L \leq n$,*

$$\mathrm{E}_{cc}(D(L)) \geq \left(1 - \epsilon^{L/(64n)}\right)^2 \frac{L}{2^{11}} .$$

Proof: Let $D(L) = l_1|\ldots|l_6$. By definition $l_1 = l_2 = l_3 - 1 = l_4 - 1 = l_5 = l$ and $l_6 > l$ where $l = \left\lfloor \frac{n}{8^{k+1}} \right\rfloor$ for integer k satisfying $\left\lfloor \frac{n}{8^k} \right\rfloor \leq L < \left\lfloor \frac{n}{8^{k-1}} \right\rfloor$. Therefore, $n = 8^{k+1} \cdot l + r$ where $r < 8^{k+1}$.

There are two cases, depending on the size of the remainder, r.

Case 1: $r \leq 4 \cdot 8^k$. In this case it will be shown that

$$\max\{\mathrm{E}_{cc}(l|l),\ \mathrm{E}_{cc}(l|l+1),\ \mathrm{E}_{cc}(l+1|l)\} \geq \left(1 - \epsilon^{l/n}\right)^2 \frac{l}{32}$$

A summary of the central ideas are as follows. The values of l and r imply that a ring of size n can be partitioned into segments of length l and $(l+1)$ in such a way that no two $(l+1)$ segments are adjacent. If all of $\mathrm{E}_{cc}(l|l)$, $\mathrm{E}_{cc}(l|l+1)$ and $\mathrm{E}_{cc}(l+1|l)$ are small and adjacent barriers are removed from the partitioned ring, then, with high probability, the traces of the additional message traffic will not intersect. Such a situation results in deadlock. So it must be concluded that at least one of $\mathrm{E}_{cc}(l|l)$, $\mathrm{E}_{cc}(l|l+1)$ or $\mathrm{E}_{cc}(l+1|l)$

is not small. However, all combinations of l and $(l+1)$ are included as adjacent lengths in the decomposition, $D(L)$. Therefore, the expected cost, over all partitions consistent with $D(L)$, of the (minimum) continuation is also not small.

Let α be any ϵ-attrition procedure for anonymous bidirectional rings of fixed size n. Let $\mathcal{P}$ be the space of deterministic processes available to α. Let x and y be two random l-processes from $\mathcal{P}^l$ and let z be a random $(l+1)$-process from $\mathcal{P}^{l+1}$. Let $\lambda = 1 - \min\left\{\Pr(CC(x|y) \leq \frac{l}{2}), \Pr(CC(x|z) \leq \frac{l}{2}), \Pr(CC(z|x) \leq \frac{l}{2})\right\}$. Then $\Pr(CC(x|y) \leq \frac{l}{2}) \geq 1 - \lambda$ and $\Pr(CC(x|z) \leq \frac{l}{2}) \geq 1 - \lambda$ and $\Pr(CC(z|x) \leq \frac{l}{2}) \geq 1 - \lambda$. Let $S_1 = \{s | s \in P^l \wedge \Pr(CC(s|y) \leq \frac{l}{2}) > 1 - \lambda^{1/2}\}$. Then $\Pr(S_1) \geq 1 - \lambda^{1/2}$ since otherwise $\Pr(CC(x|y) \leq \frac{l}{2}) < (1 - \lambda^{1/2}) + \lambda^{1/2}(1 - \lambda^{1/2}) = 1 - \lambda$. Similarly, let:

$$S_2 = \{s | s \in \mathcal{P}^l \wedge \Pr(CC(y|s) \leq \frac{l}{2}) > 1 - \lambda^{1/2}\}$$

$$S_3 = \{s | s \in \mathcal{P}^l \wedge \Pr(CC(s|z) \leq \frac{l}{2}) > 1 - \lambda^{1/2}\}$$

$$S_4 = \{s | s \in \mathcal{P}^l \wedge \Pr(CC(z|s) \leq \frac{l}{2}) > 1 - \lambda^{1/2}\}$$

Then, in the same way, $\Pr(S_i) \geq 1 - \lambda^{1/2}$ for $i = 2, 3$ and 4. Let $C = S_1 \cap S_2 \cap S_3 \cap S_4$. Then $\Pr(C) \geq 1 - 4\lambda^{1/2}$.

In summary, with high probability $(1 - \lambda^{1/2})$ a randomly chosen l-process in C will combine on either side with a randomly chosen l-process or $(l+1)$-process to produce a partition with a small (less than $l/2$) cost of continuation.

Let $d = 4 \cdot 8^k$. Consider the class of rings, B, with length n, defined by:

$$B = \left\{ x_1, \ldots, x_{2d} \mid \begin{array}{l} x_i \in C \text{ for } i = 1, 3, \ldots, 2d-1, \\ x_i \in \mathcal{P}^{l+1} \text{ for } i = 2, 4, \ldots, 2r, \\ x_i \in \mathcal{P}^l \text{ for } i = 2r+2, , \ldots, 2d, \\ \text{and } CC(x_i|x_{i+1}) \leq \frac{l}{2} \text{ for } 1 \leq i < 2d, \text{ and } CC(x_{2d}|x_1) \leq \frac{l}{2} \end{array} \right\}$$

Then $\Pr(B)$ in the product space $\mathcal{P}^n$ can be bounded as follows. Since a random l-process is in C with probability at least $1 - 4\lambda^{1/2}$, all x_i, for $i = 1, 3, \ldots, 2d-1$, are in C with

probability at least $(1-4\lambda^{1/2})^d$. Given $x_i \in C$ for $i = 1,3,\ldots,2d-1$, $CC(x_i|x_{i+1}) \leq \frac{l}{2}$ with probability at least $1-\lambda^{1/2}$ and $CC(x_{i+1}|x_{i+2}) \leq \frac{l}{2}$ with probability at least $1-\lambda^{1/2}$. Hence, for a fixed $i = 2,4,\ldots,2d$, the conditions on B are met with probability at least $1-2\lambda^{1/2}$. Hence, $\Pr(B) \geq (1-4\lambda^{1/2})^d(1-2\lambda^{1/2})^d > (1-4\lambda^{1/2})^{\lfloor n/l \rfloor}$.

For every process sequence $\Pi = x_1,\ldots,x_{2d} \in B$, schedule Π with the minimum extension $\mathcal{S}'$ of $\mathcal{S}^*(x_1|\ldots|x_{2d})$. Since $CC(x_i|x_j)$ is at most $l/2$ additional messages, and segments in the partition are of length at least l, it is impossible for messages generated by the removal of any pair of barriers to interact. Hence, after removal of all barriers there are at most $(l/2)(n/l)$ additional messages before all message traffic ceases. Since envelope-attrition is required to retain one message envelope, elements of B produce erroneous ϵ-attrition computations under scheduler $\mathcal{S}'$.

Since deadlock occurs with probability at most ϵ, $\Pr(B) \leq \epsilon$, which implies that $(1-4\lambda^{1/2})^{\lfloor n/l \rfloor} < \epsilon$. Thus, $\lambda > (1-\epsilon^{l/n})^2/16$. But from the definition of λ, either $\Pr(CC(x|y) \leq \frac{l}{2}) = 1-\lambda$ or $\Pr(CC(x|z) \leq \frac{l}{2}) = 1-\lambda$ or $\Pr(CC(z|x) \leq \frac{l}{2}) = 1-\lambda$. Hence, either $\mathrm{E}_{cc}(l|l) = \mathrm{E}(CC(x|y)) \geq \lambda \cdot \frac{l}{2} > (1-\epsilon^{l/n})^2 \frac{l}{32}$ or $\mathrm{E}_{cc}(l|l+1) > (1-\epsilon^{l/n})^2 \frac{l}{32}$ or $\mathrm{E}_{cc}(l+1|l) > (1-\epsilon^{l/n})^2 \frac{l}{32}$.

The decomposition $D(L)$ contains every combination of l and $(l+1)$ as adjacent integers. Furthermore, if a continuation of the whole partition behaves differently than the combination of the continuations on each of the adjacent segments of the partition, then there must have been interaction between these adjacent pieces. This alone would require more than $l/2$ messages. Therefore, the cost of continuation of a partition consistent with $D(L)$ must be at least the maximum of the cost of continuation on the segments composed of adjacent pairs in the partition. Thus, $\mathrm{E}_{cc}(l_1|\ldots|l_6) \geq (1-\epsilon^{l/n})^2 \frac{l}{32}$. But $l = \left\lfloor \frac{n}{8^{k+1}} \right\rfloor$ and $L < \left\lfloor \frac{n}{8^{k-1}} \right\rfloor$ implying $\frac{L}{64} \leq l$. Hence $\mathrm{E}_{cc}(l_1|\ldots|l_6) \geq (1-\epsilon^{L/(64n)})^2 \frac{L}{2^{11}}$.

Case 2: $r > 4 \cdot 8^k$. In this case,

$$\max\{\mathrm{E}_{cc}(l+1|l+1),\ \mathrm{E}_{cc}(l|l+1),\ \mathrm{E}_{cc}(l+1|l)\} \geq \left(1-\epsilon^{l/n}\right)^2 \frac{l}{32}$$

The proof of this case mimics case 1, and is therefore omitted. ∎

Theorem 7.5 *Every ϵ-attrition procedure for anonymous rings of fixed size n has expected message complexity $\Omega(n \min\{\log n, \log\log(1/\epsilon)\})$ on rings of size n.*

Proof: Let α be an ϵ-attrition procedure for rings of fixed size n and let $\mathcal{P}$ be the probability space of processes available to α.

Let $\text{cost}_{\mathcal{S}^*}(\Pi)$ be the total number of messages sent by scheduled process sequence $\mathcal{S}^*(\Pi)$ and define $\text{E}_{msg}(L) = \text{E}(\text{cost}_{\mathcal{S}^*}(\Pi)|\Pi \in \mathcal{P}^L)$. Let $\Pi \in \mathcal{P}^L$ and recall that $\delta(\Pi)$ is the partition of Π consistent with $D(L)$. Then

$$\text{cost}_{\mathcal{S}^*}(\Pi) = CC(\delta(\Pi)) + \sum_{i=1}^{6} \text{cost}_{\mathcal{S}^*}(\Pi_i)$$

implies

$$\text{E}_{msg}(L) = \text{E}\left(CC(\delta(\Pi))\right) + \sum_{i=1}^{6} \text{E}_{msg}(l_i)$$

By lemma 7.4, $\text{E}\left(CC(\delta(\Pi))\right)$ is at least $\frac{L}{2^{11}}(1-\epsilon^{L/(64n)})^2$. Therefore

$$\text{E}_{msg}(L) \;\geq\; \frac{L}{2^{11}} \cdot (1-\epsilon^{L/(64n)})^2 + \sum_{i=1}^{6} \text{E}_{msg}(l_i)$$

where $\sum_{i=1}^{6} l_i = L$ and $l_i \geq \frac{L}{64}$.

Claim:

$$\text{E}_{msg}(L) \;\geq\; \frac{L}{2^{11}} \sum_{i=1}^{\log_{64} L} (1-\epsilon^{L/(64^i n)})^2.$$

Proof of claim: The basis is clear so assume the inductive hypothesis:

$$\text{E}_{msg}(l) \;\geq\; \frac{l}{2^{11}} \sum_{i=1}^{\log_{64} l} (1-\epsilon^{l/(64^i n)})^2 \tag{7.1}$$

for all $l < L$. Then:

$$\begin{aligned}
\text{E}_{msg}(L) \;&\geq\; \frac{L}{2^{11}} \cdot (1-\epsilon^{L/(64n)})^2 + \sum_{i=1}^{6} \text{E}_{msg}(l_i) \\
&\geq\; \frac{L}{2^{11}} \cdot (1-\epsilon^{L/(64n)})^2 + \sum_{i=1}^{6} \frac{l_i}{2^{11}} \sum_{j=1}^{\log_{64} l_i} (1-\epsilon^{l_i/(64^j n)})^2 \qquad \text{(by 7.1)} \\
&\geq\; \frac{L}{2^{11}} \cdot (1-\epsilon^{L/(64n)})^2 + \sum_{i=1}^{6} \frac{l_i}{2^{11}} \sum_{j=1}^{\log_{64}(L/64)} (1-\epsilon^{L/(64^{j+1} n)})^2
\end{aligned}$$

$$\geq \frac{L}{2^{11}} \cdot (1-\epsilon^{L/(64n)})^2 + \frac{L}{2^{11}} \sum_{j=2}^{\log_{64} L} (1-\epsilon^{L/(64^j n)})^2$$

$$= \frac{L}{2^{11}} \sum_{j=1}^{\log_{64} L} (1-\epsilon^{L/(64^j n)})^2$$

So the claim holds.

Let $\mathcal{R} = \pi_1, \ldots, \pi_n$ be a random element of $\mathcal{P}^n$. Place a barrier between π_1 and π_n and consider the computation of α on segment $\pi_1, \ldots, \pi_n$ under schedule $\mathcal{S}^*$. Then:

$$\mathrm{E}_{msg}(n) \geq \frac{n}{2^{11}} \sum_{j=1}^{\log_{64} n} (1-\epsilon^{64^{-j}})^2$$

Notice that $(1-\epsilon^{64^{-x}})^2 \geq \frac{1}{4}$ as long as $x \leq \log_{64} \log(1/\epsilon) = (\log\log(1/\epsilon))/6$. Therefore:

$$\begin{aligned} \mathrm{E}_{msg}(n) &\geq \frac{n}{2^{11}} \sum_{i=1}^{\min\left\{\log_{64} n, \frac{1}{6}\log\log(1/\epsilon)\right\}} \left(1-\epsilon^{64^{-i}}\right)^2 \\ &\geq \frac{n}{2^{11}} \frac{1}{4} \min\left\{\log_{64} n, \frac{\log\log(1/\epsilon)}{6}\right\} \\ &= \Omega\left(n \min\left\{\log n, \log\log\frac{1}{\epsilon}\right\}\right) . \end{aligned}$$

■

7.3 Monte Carlo Complexity of Natural Functions

The nondeadlocking attrition procedure, γ, of section 2.2 has complexity $O(n \log c)$ expected messages when there are $c \geq 1$ initial contenders (lemma 2.1). As shown below, this attrition can be converted to an ϵ-attrition procedure, β, with expected bit complexity $O(n \min\{\log n, \log\log(1/\epsilon)\})$, when there are n initial envelopes. This shows that the $\Omega(n \min\{\log n, \log\log(1/\epsilon)\})$ expected messages bound in theorem 7.5 is tight to within a constant factor. Let $\lambda = \min\{n, \ln(1/\epsilon)\}$. Processors first choose to be contenders with probability λ/n and the contenders run γ. Then λ contenders are expected. Let c denote the number of actual contenders and let $\mathrm{complexity}_\beta$ denote the complexity of β.

$$\mathrm{E}(\mathrm{complexity}_\beta) = \sum_{c=1}^{n} \mathrm{E}(\mathrm{complexity}_\beta | c) \cdot \Pr(c)$$

$$\begin{aligned} &= \sum_{c=1}^{n} (n \log c) \cdot \Pr(c) \\ &\leq n \log \left(\sum_{c=1}^{n} c \cdot \Pr(c) \right) \text{ by the convexity of log} \\ &= n \log \mathrm{E}(c) \\ &= n \log \lambda \end{aligned}$$

Thus, β has the desired complexity. The only way that β can deadlock is if $\ln(1/\epsilon) < n$ and no processor chooses to be a contender. This happens with probability $(1 - \lambda/n)^n \leq e^{-\lambda} = \epsilon$.

The $\Omega(n \min\{\log n, \log\log(1/\epsilon)\})$ lower bound on the expected message complexity of ϵ-attrition implies the same lower bound on Monte Carlo leader election, because envelope-attrition reduces trivially to leader election. This lower bound is tight. Section 2.4 describes how a Las Vegas leader election algorithm can be assembled from nondeadlocking attrition and deterministic solitude detection. Analogously, ϵ-attrition and Monte Carlo solitude detection can be combined into a Monte Carlo leader election algorithm. As was shown in [4], Monte Carlo solitude detection has complexity $\Theta\left(n \min\left\{\sqrt{\log n}, \sqrt{\log\log(1/\epsilon)} + \log \nu(n), \log\log(1/\epsilon)\right\}\right)$ expected bits on rings of known size, where $\nu(n)$ is the smallest nondivisor of n. Thus:

Theorem 7.6 *There is a Monte Carlo leader election algorithm that errs with probability at most ϵ and has complexity $O(n \min\{\log n, \log\log(1/\epsilon)\})$ expected bits.*

The preceding discussion illustrates that envelope-attrition is an essential and dominant part of leader election in the sense that the complexities of the two problems are equivalent (even in the probabilistic case). Envelope-attrition is similarly related to other natural problems.

In lemma 6.4, nondeadlocking envelope-attrition is reduced to a Las Vegas algorithm for AND. The following is a generalization of this reduction, which illustrates that the ϵ-attrition bound extends to computing AND with probability of error at most ϵ on rings of known size.

Theorem 7.7 *The expected complexity of every Monte Carlo algorithm that with confidence at least* $1-\epsilon$ *computes AND on a ring of fixed size* n *is* $\Omega(n \min\{\log n, \log\log(1/\epsilon)\})$ *messages.*

Proof: Let α be a Monte Carlo algorithm for AND which errs with probability at most ϵ. Let $f(n,\epsilon)$ be the expected message complexity of α. Let γ be the attrition procedure of section 2.2. On a ring of size n, the expected complexity of γ is $O(n \log c)$ messages, where c is the actual number of contenders (not necessarily known). Let $\lambda = \min\{\ln n, \ln\ln(1/\epsilon)\}$. Each processor on a ring executes the following:

```
Procedure ATTRITION2:
1. generate a random bit, myflip ∈ {0,1} such that Pr(0) = λ/n;
2. IF myflip = 0 THEN
      become a contender and initiate γ;
      henceforth participate in γ and discard all α messages
   ELSE
      initiate α;
      Participate in α only as long as no γ message arrives;
      Upon receipt of a γ message, participate in γ as a
      non-contender and discard all subsequent α messages;
      IF α confirms ''all 1's'' THEN go to step 1.
```

Step 2 performs ϵ-attrition on an expected small number of contenders. In the event that there were no contenders, the processors are alerted to try again. Thus, *ATTRITION2* is an ϵ-attrition algorithm.

Error: The only way that the ϵ-attrition procedure, *ATTRITION2*, deadlocks is if all processors flip 1 and the AND algorithm α fails to confirm all 1's. Therefore, the probability of deadlock of *ATTRITION2* is at most $\epsilon \sum_{i=1}^{\infty} \left(\left(1-\frac{\lambda}{n}\right)^n\right)^i \leq \frac{\epsilon}{e^\lambda - 1} \leq \epsilon$ as long as $\epsilon \leq 1/e$. So $\epsilon < 0.367$ suffices.

Complexity: Let random variable C be the number of processors with $myflip = 1$. Denote the number of messages sent by *ATTRITION2* by complexity_{ATT2}. Then the expected complexity of *ATTRITION2* is given by

$$\begin{aligned}
\mathrm{E}(\text{complexity}_{ATT2}) &= \mathrm{E}(\text{complexity}_{ATT2}|\text{all 1's})\Pr(\text{all 1's}) + \\
&\quad \mathrm{E}(\text{complexity}_{ATT2}|\text{at least one 0})\Pr(\text{at least one 0}) \\
&\leq (f(n,\epsilon) + \mathrm{E}(\text{complexity}_{ATT2}))\left(1-\frac{\lambda}{n}\right)^n + \\
&\quad (f(n,\epsilon) + \mathrm{E}(n\log C))\left(1-\left(1-\frac{\lambda}{n}\right)^n\right)
\end{aligned}$$

Therefore

$$\begin{aligned}
\mathrm{E}(\text{complexity}_{ATT2}) &\leq \frac{f(n,\epsilon) + n\log\lambda\left(1-e^{-\lambda}\right)}{1-e^{-\lambda}} \\
&< 2f(n,\epsilon) + n\min\left\{\log\log n, \log\log\log\frac{1}{\epsilon}\right\} \\
&\quad (\text{if } n > 2 \text{ and } \epsilon < 0.135)
\end{aligned}$$

Since *ATTRITION2* is an ϵ-attrition algorithm, the expected complexity of *ATTRITION2* is $\Omega(n\min\{\log n,\ \log\log(1/\epsilon)\})$ bits. Hence, the expected complexity of $f(n,\epsilon)$ is $\Omega(n\min\{\log n,\ \log\log(1/\epsilon)\})$ bits. ■

It is easily verified that ϵ-attrition also reduces to other functions such as OR or PARITY, and lower bounds for SUM are inherited from PARITY. Furthermore, the lower bounds are tight because AND (and similarly OR and PARITY) can be computed on a ring of size n by expending an additional $O(n)$ bits after electing a leader. Therefore:

Corollary 7.8 *The expected complexity of computing AND (or OR or PARITY) with confidence at least* $1-\epsilon$ *on a ring of fixed size* n *is* $\Theta(n\min\{\log n, \log\log(1/\epsilon)\})$ *messages and bits.*

The leader election of theorem 7.6 that is used to achieve the lower bounds above is composed of the Monte Carlo solitude detection algorithm in [4] and ϵ-attrition. The Monte Carlo solitude detection algorithm errs only by occasionally terminating with the wrong answer and only one-sided error is possible. On the other hand, the ϵ-attrition procedure errs only by deadlocking. Thus, the leader election algorithm exhibits two types of possible error — deadlock and termination with an incorrect conclusion.

Alternative algorithms for leader election and thus for AND and other natural functions for rings of fixed size n can be constructed from the algorithm *SD* in section 5.3 and ϵ-attrition. These algorithms err only by deadlocking. This approach is illustrated here for AND; it is a variation of algorithm *SIMPLE-LE* in section 2.4. Let $\lambda = \min\{n, \log(1/\epsilon)\}$.

```
Algorithm AND3:
   generate a random bit, myflip ∈ {0,1} such that Pr(0) = λ/n;
   IF myflip = 0 THEN contender ←true;
   WHILE contender AND NOT leader DO
      log λ rounds of attrition;
      IF contender THEN
         initiate algorithm SD
         IF solitude is confirmed THEN leader ←t
   IF leader THEN accumulate the AND bit.
```

Each execution of solitude detection requires $O(n\sqrt{\log n})$ expected bits. Each execution of the attrition step requires $n \log \lambda$ bits. Since λ contenders are expected initially, a constant number of passes through the WHILE loop are expected, and the repeated passes through the WHILE loop do not increase the order of the expected complexity over that of a single pass. Thus, *AND3* is a Monte Carlo algorithm which errs only by deadlocking and has an expected complexity of $O(n\sqrt{\log n} + n \log\log(1/\epsilon))$ bits for $\epsilon \geq 2^{-n}$ where ϵ is the probability of deadlock.

Chapter 8

Conclusions

8.1 Summary of Contributions

Many fundamental issues that affect the complexity of computing on a ring have been addressed in the previous chapters. One object of this section is to summarize these issues and point to specific examples which illustrate them. Principal among these factors is the effect of randomization on distributed computing. Numerous other factors are also summarized. Another goal is to isolate some general techniques from particular results. The advantages of the general model of chapter 4, and the most significant results achieved using this model are reiterated.

8.1.1 Impact of randomization

New efficient randomized algorithms for fundamental distributed computing problems on rings have been described. The following list summarizes the positive contributions of randomization to the solutions of some of these problems.

1. Some of the Las Vegas algorithms provide solutions to problems that have no deterministic solution. For example, each of the problems, leader election, orientation, SUM, and PARITY, on anonymous rings of size $n \in [N/2+1, N]$ can be solved by a

Las Vegas algorithm, whereas no deterministic solutions exist for this class of rings.

2. For some problems, though deterministic solutions exist, Las Vegas solutions improve significantly upon the complexity of any deterministic solution. Deterministic solutions of AND and OR on an anonymous ring of size $n \in [N/2, N]$ have complexity $\Omega(n^2)$ messages compared to $O(n \log n)$ expected bits for Las Vegas algorithms. In fact, all functions on rings of size $n \in [N/2+1, N]$ can be computed by a Las Vegas algorithm using at most $O(n \log n)$ expected messages because leader election has this complexity, whereas many require $\Omega(n^2)$ messages in the deterministic model. A nontrivial function is described in section 5.3 which can be computed in $O(n\sqrt{\log n})$ expected bits on rings of fixed size n, but no deterministic algorithm for a nontrivial function has complexity less than $\Omega(n \log n)$ bits ([21]).

3. Randomization makes complexity dependent on coin flips as well as on the processors' inputs. This dependence may eliminate the need to design an elaborate algorithm which guards against expensive computations that arise from specific inputs. As a consequence, efficient Las Vegas algorithms are often also simple. Many of the algorithms in the previous chapter have a simple structure. Most employ attrition, which is a natural technique if randomization is available.

4. The possibility of probabilistic solutions can be entertained in models that support randomization. For many natural problems, when a solution with confidence of $1 - \epsilon$ is sufficient, a Monte Carlo algorithm exists that has lower complexity than the corresponding Las Vegas one (cf. theorem 7.6). The solitude verification results in [4] demonstrate that even when there is no algorithm that is correct with certainty, there are Monte Carlo algorithms that are almost certainly correct.

5. The message complexity of Las Vegas solutions on anonymous rings frequently compares favourably with that of deterministic solutions for rings with distinct identifiers. Besides not requiring identifiers, the Las Vegas solutions are frequently achieved using $O(1)$ length messages as opposed to $O(\log n)$ length messages for deterministic solutions. The Las Vegas leader election algorithm, *LE*, in chapter 2 on page 22,

has an expected message complexity of the same order as the average complexity of deterministic leader election on an asynchronous ring with distinct identifiers. Moreover, the complexity is achieved using messages of constant length as opposed to the messages of size about $\log n$ bits for the deterministic algorithms.

These examples of the positive effect of randomization support a central thesis of this research: that randomization is an effective tool for the design of efficient distributed algorithms. Some of the reasons for this effectiveness are discussed in chapter 1.

Randomization is by no means a panacea, however. Consider the function:

$$f(i_1, \ldots, i_{2n}) = \begin{cases} 1 & \text{if } i_1 \ldots i_n = i_{n+1} \ldots i_{2n} \\ 0 & \text{otherwise} \end{cases}$$

There is an obvious $O(n^2)$ bits deterministic algorithm for evaluating f on rings of known (and even) size. Each processor sends its input and all messages are forwarded n times. A simple argument, adapted from [32], can be used to show that there is some input for which $\Omega(n^2)$ bits are required to certify a function value of 1. Suppose there is some nondeterministic algorithm α that has a correct computation for all input strings xx where the length of x is $n/2$. For each such input, the corresponding correct computation must determine that each input bit matches the one diametrically opposite. There are on the order of 2^n possible strings y such that $f(y)$ is 1. By an adaptation of well known crossing sequence arguments for Turing machines, there is some input for which there must be on the order of $\log(2^n)$ bits sent across each of $n/2$ links. Thus, the n comparisons require n^2 bits of communication for this input. Hence, even nondeterministic algorithms cannot significantly reduce the *bit* complexity of evaluating f. (Notice, however, that the $\Omega(n^2)$ *messages* deterministic complexity of f can be reduced to $O(n \log n)$ *expected messages* in a randomized model by first executing Las Vegas leader election.)

8.1.2 Model

Las Vegas algorithms presented in this thesis are accompanied by matching lower bounds. To derive such bounds requires a precise model of the communication that occurs during

a distributed computation. Chapter 4 contributes a unified collection of models that captures the essential characteristics of a spectrum of distributed algorithms — those that are error free (deterministic, Las Vegas, and nondeterministic), and those that err with small probability (Monte Carlo and nondeterministic/probabilistic). The unification helps to clarify the essential differences between the progressively more general notions of a distributed algorithm. The bounds frequently vary depending on the type of algorithm being investigated, thus underscoring the fact that these differences are more than notational.

Various parameters, in addition to the type of algorithm being considered, can be accommodated within the general framework of the model. These include the class of rings for which the algorithm is required to work, the presence or absence of identifiers, the domain of input values for the algorithm, and the termination requirements of the algorithm. The lower bounds are sensitive to assumptions concerning these parameters. The fact that the model reveals this sensitivity, attests to the appropriateness of the model.

8.1.3 General techniques

Techniques for designing randomized algorithms and for proving randomized lower bounds can be distilled from particular results.

Many of the algorithms can be viewed as combinations of two techniques — coin flipping and counting — and variations on these techniques. Coin flipping is used to break symmetry while counting facilitates termination. This is first seen in the leader election algorithm (algorithm *LE* on page 22) where attrition of contenders is achieved via exchanges of random coin tosses, and solitude is verified by sending a counter to measure the gaps between remaining contenders.

Variations of straightforward counting occur in subsequent algorithms. Counting modulo a small number and counting up to a threshold are combined in the algorithm for solitude detection when ring size is fixed (algorithm *SD* on page 71). Number theoretic properties of the ring size are exploited to derive enough information from these two cheaper kinds of counting to determine solitude. In algorithm *AND2* in chapter 6 on page 79, the

values of adjacent gap counters are compared to assure that all gaps (likely only one) are the same size. Probabilistic extensions of these counting techniques are developed in [3] and [4]. Ordinary and threshold counting are replaced by probabilistic counting in which processors increment a counter with a fixed probability less than one rather than with certainty. In another variation (similar to a scheme used in [16]) a collection of processors repeat the experiment of randomly choosing from $\{0, 1\}$ such that 1 is chosen with low but increasing probability until at least one processor chooses 1. These schemes yield an estimate on the number of participating processors. In all of these examples, the purpose of counting is to gather enough information to assure correct termination.

Reducing the number of message envelopes is central to many of the algorithms. It is always achieved by a simple exchange of strings of random bits. Both nondeadlocking and ϵ-attrition proceed similarly, the only difference being that by risking deadlock the ϵ-attrition version can significantly reduce the initial number of contenders without incurring any communication cost. After this reduction, ϵ-attrition is the same as nondeadlocking attrition on the smaller set of contenders.

Lower bound techniques are of two distinct kinds, one especially suited for nondeterministic or best case bit complexity, the other for expected message complexity. Best case bit complexity proofs begin with the assumption that there is some computation that has low complexity. Familiar combinatorial arguments are used to conclude that some processors have identical histories. The repeated histories provide the endpoints of segments that can be removed from the ring without affecting the histories of the remaining processes (shrinking). The reduced computation is replicated and, if necessary, an additional piece is spliced into the ring to produce a new ring and computation of the required length. After shrinking, replicating and splicing, what results is a feasible but erroneous computation of the algorithm. Hence, the assumption of low complexity must be incorrect.

Proving lower bounds on the expected message complexity of algorithms that admit efficient best cases requires techniques that distinguish between best case and average case computations. Thus, the techniques must account for probabilities of computations. The idea is to impose a scheduler on the computation in such a way that the computation is

divided into disjoint parts that can be individually analysed. The ring is partitioned into segments by inserting blocks between processes. Different levels are associated with the blocks; and the blocks are removed one level at a time as the computation progresses. As each set of blocks is removed, enough additional communication must occur to ensure that the computation does not deadlock. Thus, a lower bound is derived for the expected amount of communication that takes place at each level. By summing over the disjoint parts of the computation, a lower bound on total expected communication is derived.

8.1.4 Issues effecting complexity

type of algorithm:

The type of algorithm (deterministic, Las Vegas, Monte Carlo, nondeterministic, or nondeterministic/probabilistic) is frequently a major factor in determining the complexity of the solution to a distributed computing problem. Differences between the complexities of deterministic and Las Vegas solutions for many problems have already been summarized in subsection 8.1.1.

The complexity of computing AND on rings of known size exhibits extreme sensitivity to the type of algorithm being considered. The complexity of AND on rings of fixed size decreases for each generalization from one of these types of algorithms to a more powerful one. The deterministic complexity of AND on rings of known size n is $\Theta(n^2)$ bits. The Las Vegas complexity is $\Theta(n \log n)$ expected bits; and the nondeterministic complexity is $\Theta(n\sqrt{\log n})$ bits. The complexity of AND in the Monte Carlo model is $\Theta(n \min\{\log n, \log\log(1/\epsilon)\})$ expected bits. A nondeterministic/probabilistic algorithm for AND, with complexity $O\left(n \min\left\{\sqrt{\log n}, \sqrt{\log\log(1/\epsilon)} + \log \nu(n), \log\log(1/\epsilon)\right\}\right)$ expected bits on rings of known size, where $\nu(n)$ is the smallest nondivisor of n, can be constructed from a combination of the $\Theta(n \log \nu(n))$ attrition of section 7.1, and the nondeterministic/probabilistic solution for solitude verification from [4].

Not all problems are as sensitive to algorithm type as is AND on rings of known size. The function f described in subsection 8.1.1 (page 110) illustrates the opposite extreme

where algorithm type has essentially no effect on bit complexity.

For other problems, the complexity decreases with a change from the deterministic to the Las Vegas model, but does not further decrease through a generalization from the Las Vegas to the nondeterministic model. This was shown for problems such as leader election, AND, and PARITY on rings of size $n \in [N/2, N]$, as well as for the function described in chapter 5 on page 74.

knowledge of ring size:

Another theme of this dissertation is the degree to which knowledge of ring size influences the inherent complexity of the solution to a given problem. Las Vegas solutions for solitude detection and for minimum nonconstant function evaluation are sensitive to exact knowledge of ring size (chapter 5). While the best case complexity of AND and (many other functions) is reduced for rings of known size, the expected complexity is insensitive to information about ring size that is more refined than to within a constant factor (chapter 7).

The range of ring sizes for which a solution even exists depends upon the type of algorithm that is required. Severe constraints on the ring size are required for deterministic solutions for SUM (n known exactly) and for Orientation (n known exactly and n odd) for anonymous rings. However, both these problems can be solved by a Las Vegas algorithm as long as ring size is known to within a factor of two.

type of termination:

Some problems exhibit a drop in complexity when only nondistributive termination is required, while for others the nondistributive termination lower bounds can be achieved with distributively terminating algorithms.

AND provides an example of a significant drop in complexity, from $\Theta(n^2)$ bits (deterministic) or $\Theta(n \log n)$ expected bits (Las Vegas) to $\Theta(n)$ bits, when the requirement is weakened from distributive to nondistributive termination. For a more elaborate and less

obvious example, see [4]. There it is shown that the complexity of solitude detection on rings of known size drops from $\Theta(n\sqrt{\log n})$ to $\Theta(n \log\log n)$ expected bits under the same relaxation of the termination condition.

permissibility of error:

It has already been observed that complexity can often be decreased if a solution that is correct with high probability rather than correct with certainty is acceptable. The natural functions discussed in chapter 7 have complexity $\Theta(n \log\log(1/\epsilon))$ expected bits if error with probability $\epsilon > 1/2^n$ can be tolerated, whereas, the complexity is $\Theta(n \log n)$ expected bits if no error is allowed. Other examples occur in [3] and [4] where solitude detection problems are shown to have lower complexity in the Monte Carlo model than in the Las Vegas Model.

kind of error:

The traditional notion of a Monte Carlo algorithm is one that errs with small probability. Typically an algorithm, α, for a decision problem may err in a number of ways. One type of error is when α terminates with the wrong decision. Alternatively, α may err by failing to terminate. Finally, α may terminate without any decision. A more restricted notion of a Monte Carlo algorithm (cf. for example [11]) assumes that a Monte Carlo algorithm always terminates with some decision in a bounded amount of time. Thus, the only error permitted is of the first kind.

In the classical single processor environment, this restriction is not an essential one. Typically a Monte Carlo algorithm that errs with probability at most ϵ can be converted to a restricted Monte Carlo algorithm that errs with probability at most ϵ by enforcing termination with a predefined conclusion for computations that exceed a sufficiently large bound on running time.

Notice that a clock is central to this conversion. In the distributed asynchronous setting there is an essential distinction in the various type of errors.

The algorithm *AND3* of page 107 errs only by deadlocking. It is correct given deadlock does not occur, and has complexity $O(n\sqrt{\log n} + n \log\log(1/\epsilon))$ expected bits. The more efficient Monte Carlo algorithm for AND which is constructed from Monte Carlo leader election (corollary 7.8) has complexity $O(n \min\{\log n, \log\log(1/\epsilon)\})$ expected bits. This algorithm errs both by deadlocking and by terminating with an incorrect answer. Admissibility of both kinds of error is essential to achieve this drop in complexity.

existence of identifiers:

Rings with identifiers have not been a significant focus of the preceding chapters. It has been observed, however, that there is some overlap of the role played by processor identifiers and by random sequences in distributed computation. The message complexity of deterministic algorithms on rings with distinct identifiers is often of the same order as that of Las Vegas algorithms on anonymous rings. The leader election algorithm for rings with distinct identifiers (chapter 2, page 23) illustrates this phenomenon. It is the relationship between these two models that accounts for the similarity in proof techniques between the average case leader election results in [14], and the expected case Las Vegas AND result described in chapter 7 on page 92.

8.1.5 Fundamental problems

It was shown in chapter 2 that leader election can be decomposed into two even more fundamental problems, solitude verification and attrition. These two subproblems of leader election continued to appear throughout the remaining chapters. The algorithms either implicitly or explicitly incorporate a solution to at least one of these two problems. Furthermore, the lower bounds are derived either by mimicking the proofs of the lower bounds for attrition and solitude verification, or by appealing directly to a reduction from one of these problems.

This is evidence of the distinguished role of these two problems. The function that achieves the lower bound for nontrivial function evaluation (chapter 5, page 74) is an

encoding of solitude verification. Hence, solitude verification can be thought of as the generic representative of the simplest nontrivial problems. Attrition, on the other hand, is pervasive. For a large class of natural problems, attrition is required of any efficient algorithm for any problem in the class.

8.1.6 Principal results

Two specific results can be isolated from the others because of their relative significance.

1. If g is any nonconstant cyclic function of n variables, then any nondeterministic algorithm for computing g on an anonymous ring of size n has complexity $\Omega(n\sqrt{\log n})$ bits of communication; and, there is a is nonconstant cyclic boolean function f, such that f can be computed by a Las Vegas algorithm in $O(n\sqrt{\log n})$ expected bits of communication on a ring of size n.

2. The expected complexity of computing AND (and a number of other natural functions) on a ring of fixed size n within the Monte Carlo model is $\Theta(n \min\{\log n, \log\log(1/\epsilon)\})$ messages and bits where ϵ is the allowable probability of error.

8.2 Further Research

There are two problems that are immediately suggested by results in this dissertation, but which remain unsolved.

1. As described in section 5.4, the extension of item 1 in subsection 8.1.6 above to include Monte Carlo algorithms is incomplete.

2. The extension of the $\Omega(n \log n)$ bits lower bound for the best case of Las Vegas algorithms that compute AND on a ring of size $n \in [N/2, N]$ to a nondeterministic lower bound remains a conjecture.

The investigation of other topics related to those already discussed has been carried to various stages of completion.

leader election in an arbitrary network:

A leader can be elected deterministically on an arbitrary network with unique identifiers using $O(e+n \log n)$ messages in the worst case, where n is the number of nodes in the network, and e is the number of edges [26]. Each message has length at least $\log n$ bits. It might be hoped that message length could be reduced to a constant by using random sequences in place of identifiers. However, it is possible that forwarding information dominates the coin toss values resulting in nonconstant length messages. Yamashita and Kameda [30,31] have characterized the anonymous networks for which deterministic leader election is possible. The Las Vegas leader election algorithm in chapter 2 exploits randomization to break symmetry. A similar exchange of coin tosses can be used reduce the number of contenders on an arbitrary network. If randomization is permitted, there are no restrictions to leader election provided the network size is known.

tradeoffs:

Recall that the Las Vegas complexity of solitude verification on ring of known size is $\Theta(n\sqrt{\log n})$ expected bits. The square root term results from minimizing the bit complexity, which can be achieved only at the expense of sending more messages. In fact, the complexity is a specific point in a tradeoff between the message and bit complexity for this problem. It appears that the square root term appearing in the complexity results for other problems on rings of known size (chapter 5, see also [4,6]) signals a similar tradeoff between message and bit complexity. Another parameter, maximum message length, introduces further tradeoffs between each of the message and bit parameters. Tradeoffs have been established for both deterministic and Las Vegas solitude verification. The extensions to the Monte Carlo model seem to be more involved.

bidirectional rings:

It is natural to enquire whether the bit complexity results for unidirectional rings extend to the bidirectional case. It appears that shrinking, replicating and splicing techniques can be adapted to bidirectional rings.

time complexity:

The *time complexity* of a Las Vegas asynchronous algorithm is the maximum, over all inputs and all schedules, of the expected number of unit time intervals before the algorithm terminates, under the assumption that messages travel each communication link in at most unit time and local processing is instantaneous. Nothing in this dissertation has addressed the time complexity of distributed computation. But the analysis of asynchronous time may be simplified on a unidirectional ring because the scheduler cannot influence computation sequences. For this reason the ring is an appealing topology on which to begin an investigation of time complexity.

References

[1] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. The bit complexity of probabilistic leader election on a unidirectional ring. In *Distributed Algorithms on Graphs*, pages 1–12, Carleton University Press, 1986. Proc. 1st International Workshop on Distributed Algorithms.

[2] K. Abrahamson, A. Adler, R. Gelbart, L. Higham, and D. Kirkpatrick. The bit complexity of randomized leader election on a ring. *SIAM Journal on Computing*, 1988. In press.

[3] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. *Probabilistic Solitude Detection I: Rings Size Known Approximately.* Technical Report 87-8, University of British Columbia, 1987. submitted for publication.

[4] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. *Probabilistic Solitude Detection II: Rings Size Known Exactly.* Technical Report 86-26, University of British Columbia, 1986. submitted for publication.

[5] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Probabilistic solitude verification on a ring. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 161–173, 1986.

[6] K. Abrahamson, A. Adler, L. Higham, and D. Kirkpatrick. Randomized function evaluation on a ring. In *Lecture Notes in Computer Science #312*, pages 324–331, Springer Verlag, 1987. Proc. 2nd International Workshop on Distributed Algorithms.

[7] Y. Afek and E. Gafni. Simple and efficient distributed algorithms for election in complete networks. In *Proc. 22nd Ann. Allerton Conf. on Communication, Control, and Computing*, pages 689–698, 1984.

[8] D. Angluin. Local and global properties in networks of processors. In *Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing*, pages 82–93, 1980.

[9] H. Attiya, N. Santoro, and S. Zaks. *From Rings to Complete Graphs — $\Theta(n \log n)$ to $\Theta(n)$ Distributed Leader Election.* Technical Report SCS-TR-109, Carleton University, 1987.

[10] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. In *Proc. 4th Annual ACM Symp. on Principles of Distributed Computing*, pages 196–203, 1985.

[11] G. Brassard and P. Bratley. *Algorithmics Theory and Practice.* Prentice-Hall, 1987. pre-publication manuscript, Université de Montréal.

[12] J. Burns. *A Formal Model for Message Passing Systems.* Technical Report TR-91, Indiana University, 1980.

[13] D. Dolev, M. Klawe, and M. Rodeh. An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle. *J. Algorithms*, 3(3):245–260, 1982.

[14] P. Duris and Z. Galil. Two lower bounds in asynchronous distributed computation (preliminary version). In *Proc. 28nd Annual Symp. on Foundations of Comput. Sci.*, pages 326–330, 1987.

[15] G. Fredrickson and N. Lynch. The impact of synchronous communication on the problem of electing a leader in a ring. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, pages 493–503, 1984.

[16] A. Greenberg and R. Ladner. Estimating the multiplicities of conflicts in multiple access channels. In *Proc. 24nd Annual Symp. on Foundations of Comput. Sci.*, pages 383–392, 1983.

[17] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. In *Proc. 22nd Annual Symp. on Foundations of Comput. Sci.*, pages 150–158, 1981.

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

[19] M. Loui, T. Matsushita, and D. West. Election in a complete network with a sense of direction. *Information Processing Letters*, 22(4):185–187, 1986.

[20] Y. Mansour and S. Zaks. On the bit complexity of distributed computations in a ring with a leader. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 151–160, 1986.

[21] S. Moran and M. Warmuth. Gap theorems for distributed computation. In *Proc. 5th Annual ACM Symp. on Principles of Distributed Computing*, pages 131–140, 1986.

[22] T. Nagell. *Introduction to Number Theory.* John Wiley and Sons Inc., New York, 1951.

[23] J. Pachl. *A Lower Bound for Probabilistic Distributed Algorithms.* Technical Report CS-85-25, University of Waterloo, Waterloo, Ontario, 1985.

[24] J. Pachl, E. Korach, and D. Rotem. Lower bounds for distributed maximum finding. *J. Assoc. Comput. Mach.*, 31(4):905–918, 1984.

[25] G. Peterson. An $O(n \log n)$ algorithm for the circular extrema problem. *ACM Trans. on Prog. Lang. and Systems*, 4(4):758–752, 1982.

[26] P. H. Robert Gallager and P. Spira. A distributed algorithm for minimum weight spanning trees. *ACM Trans. on Prog. Lang. and Systems*, 5(1):66–77, 1983.

[27] J. Sack, N. Santoro, and J. Urrutia. *$O(n)$ Election Algorithms in Complete Graphs With Sense of Direction.* Technical Report SCS-TR-49, Carleton University , Ottawa, Ontario, 1984.

[28] N. Santoro. Sense of direction, topological awareness, and communication complexity. *SIGACT News*, 16(2):50–56, 1984.

[29] V. Syrotiuk and J. Pachl. *Average Complexity of a Distributed Orientation Algorithm.* Technical Report CS-87-23, University of Waterloo, Waterloo, Ontario, 1987.

[30] M. Yamashita and T. Kameda. *Computing Functions on an Anonymous Network.* Technical Report LCCR-TR-87-16, Simon Fraser University, Burnaby, British Columbia, 1987.

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

[32] A. Yao. Some complexity questions related to distributive computing. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, pages 209–213, 1979.