Yale University
Department of Computer Science

The Wakeup Problem

(EXTENDED ABSTRACT)

M.J. Fischer S. Moran S. Rudich G. Taubenfeld

YALEU/DCS/TR-771
March 1990

This paper will appear in the proceedings of the Twenty-Second Annual Symposium on Theory of Computing (STOC), Baltimore, Maryland, May 1990.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR 771	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): THE WAKEUP PROBLEM	5. TYPE OF REPORT & PERIOD COVERED: Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Michael J. Fischer, Yale Shlomo Moran, Technion Steven Rudich, Carnegie Mellon Gadi Taubenfeld, Yale	8. CONTRACT OR GRANT NUMBER(s): ONR N00014-89-J-1980 (see p.1 of report for other support)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Yale University Dept. of Computer Science P.O. Box 2158 Yale Station New Haven, CT 06520-2158	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: March 1990
	13. NUMBER OF PAGES: 11
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research 800 North Quincy Street Arlington, VA 22217	15. SECURITY CLASS. (of this report): unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for pubic release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

wakeup problem
consensus
leader election
shared memory
crash failures
theory of knowledge
resiliency

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We study a new problem ,the wakeup problem that seems to be very fundamental in distributed computating. We present efficient solutions to the problem, and show how these solution can be used to solve the consensus problem, the leader election problem, and other related problems. The main question we try to answer is, how much memory is needed to solve the wakeup problem? We assume a model that captures important properties of real systems that have been largely ignored by previous work on cooperative problems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The Wakeup Problem

(EXTENDED ABSTRACT)

Michael J. Fischer* Shlomo Moran† Steven Rudich‡ Gadi Taubenfeld*

Abstract

We study a new problem, the *wakeup problem*, that seems to be very fundamental in distributed computing. We present efficient solutions to the problem and show how these solutions can be used to solve the consensus problem, the leader election problem, and other related problems. The main question we try to answer is, how much memory is needed to solve the wakeup problem? We assume a model that captures important properties of real systems that have been largely ignored by previous work on cooperative problems.

1 Introduction

1.1 The Wakeup Problem

The *wakeup problem* is a deceptively simple new problem that seems to be very fundamental to distributed computing. The goal is to design a t-resilient protocol for n asynchronous processes in a shared memory environment such that at least p processes eventually learn that at least τ processes have waked up and begun participating in the protocol. Put another way, the wakeup problem with parameters n, t, τ and p is to find a protocol such that in any fair run of n processes with at most t failures, at least p processes eventually *know* that at least τ processes have taken at least one step in the past. The only kind of failures we consider are crash failures, in which a process may become faulty at any time during its execution, and when it fails, it simply stops participating in the protocol.

In the wakeup problem, it is known a priori by all processes that at least $n - t$ processes will eventually wake up. The goal is simply to have a point in time at which the fact that at least τ processes have already waked up is *known* to p processes. It is not required that this time be the earliest possible, and faulty processes are included in the counts of processes that have waked up and that know about that fact. Note that in a solution to the wakeup problem, at least $p - t$ correct processes eventually learn that at least $\tau - t$ correct processes are awake and participating in the protocol.

The significance of this problem is two-fold. First, it seems generally useful to have a protocol such that after a crash of the network or after a malicious attack, the remaining correct processes can figure out if sufficiently many other processes remain active to carry out a given task. Second, a solution to this problem is a useful building block for solving other important problems (cf. section 6).

1.2 A New Model

Much work to date on fault-tolerant parallel and distributed systems has been generous of the class of faults considered but rather strict in the requirements on the system itself. Problems are usually studied in an underlying model that is fully synchronous, provides each process with a unique name that is known to all other processes, and is initialized to a known state at time zero. We argue that none of these assumptions is realistic in today's computer networks, and achieving them even within a single parallel computer is becoming increasingly difficult and costly. Large systems do not run off of a single clock and hence are not synchronous. Providing processes with unique id's is costly and difficult and greatly complicates reconfiguring the system. Finally, simultaneously resetting all of the computers and communication channels in a large network to a known initial state is virtually impossible and would rarely be done even if it were possible because of the large destructive effects it would have on ongoing activities.

Our new model of computation makes none of these

*Computer Science Department, Yale University, New Haven, CT 06520.

†Computer Science Department, Technion, Haifa 32000, Israel.

‡Computer Science Department Carnegie Mellon University, Pittsburgh, PA 15213.

This work was supported in part by ONR contract N00014-89-J-1980, by the National Science Foundation under grant CCR-8405478, by the Hebrew Technical Institute scholarship, by the Technion V.P.R. Funds - Wellner Research Fund, and by the Foundation for Research in Electronics, Computers and Communications, administrated by the Israel Academy of Sciences and Humanities.

assumptions. It consists of a fully asynchronous collection of n identical anonymous deterministic processes that communicate via a *single* finite sized shared register which is initially in an arbitrary unknown state. Access to the shared register is via atomic "test-and-set" instructions which, in a single indivisible step, read the value in the register and then write a new value that can depend on the value just read.

Assuming an arbitrary unknown initial state relates to the notion of self-stablizing systems defined by Dijkstra [8]. However, Dijkstra considers only nonterminating control problems such as the mutual exclusion problem, whereas we show how to solve decision problems such as the wakeup, consensus and leader election problems, in which a process makes an irrevocable decision after a finite number of steps.

Before proceeding, we should address two possible criticisms of shared memory models in general and our model in particular. First, most computers implement only reads and writes to memory, so why do we consider atomic test-and-set instructions? One answer is that large parallel systems access shared memory through a communication network which may well possess independent processing power that enables it to implement more powerful primitives than just simple reads and writes. Indeed, such machines have been seriously proposed [23, 44]. Another answer is that part of our interest is in exploring the boundary between what can and cannot be done, and a proof of impossibility for a machine with test-and-set access to memory shows *a fortiori* the corresponding impossibility for the weaker read/write model.

A second possible criticism is that real distributed systems are built around the message-passing paradigm and that shared memory models are unrealistic for large systems. Again we have several possible answers. First, the premise may not be correct. Experience is showing that message-passing systems are difficult to program, so increasing attention is being paid to implementing shared memory models, either in hardware (e.g. the Fluent machine [45]) or in software (e.g. the Linda system [5]). Second, message-passing systems are themselves an abstraction that may not accurately reflect the realities of the underlying hardware. For example, message-passing systems typically assume infinite buffers for incoming messages, yet nothing is infinite in a real system, and indeed overflow of the message buffer is one kind of fault to which real systems are subject. It is difficult to see how to study a kind of fault which is assumed away by the model. Finally, at the lowest level, communication hardware looks very much like shared memory. For example, a wire from one process to another can be thought of as a binary shared register which the first process can write (by injecting a voltage) and the second process can read (by sensing the voltage).

1.3 Space Complexity Results

The main question we try to answer is, how many values v for the shared register are necessary and sufficient to solve the wakeup problem? The answer both gives a measure of the communication-space complexity of the problem and also provides a way of assessing the cost of achieving reliability. We give a brief overview of our results below.

1.3.1 Fault-Free Solutions

First we examine what can be done in the absence of faults (i.e., $t = 0$). We present a solution to the wakeup problem in which one process learns that all other processes are awake (i.e., $p = 1$ and $\tau = n$), and it uses a single 4-valued register (i.e., $v = 4$). The protocol for achieving this is quite subtle and surprising. It can also be modified to solve the leader election problem. Based on this protocol, we construct a fault-free protocol that reaches consensus on one out of k possible values using a 5-valued register. Finally, we show that there is no fault-free solution to the wakeup problem with only two values (i.e., one bit) when $\tau \geq 3$.

1.3.2 Fault-Tolerant Solutions: Upper Bounds

We start by showing that the fault-free solution which uses a single 4-valued register, mentioned in the previous section, can actually tolerate t failures for any $\tau \leq ((2n-2)/(2t+1)+1)/2$. Using many copies of this protocol, we construct a protocol with $v = 8^{t+1}$ that tolerates t faults when $\tau \leq n - t$. Thus, if t is a constant, then a constant sized shared memory is sufficient, independent of n. However, the constant grows exponentially with t. An easy protocol exists with $v = n$ that works for any t and $\tau \leq n-t$. This means that the above exponential result is only of interest for $t \ll \log n$. Finally, we show that for any $t < n/2$, there is a t-resilient solution to the wakeup problem for any $\tau \leq \lfloor n/2 \rfloor + 1$, using a single $O(t)$-valued register.

1.3.3 Fault-Tolerant Solutions: A Lower Bound

We prove that for any protocol P that solves the wakeup problem for parameters n, t and τ, the number of shared memory values used by P is at least W^{α}, where $W = (t\sqrt{t} - t)/(n - t)$ and $\alpha = 1/(\log_2(\frac{n-t}{t+2} + 3))$. The proof is quite intricate and involves showing for any protocol with too few memory values that there is a run in which $n - t$ processes wake up and do not fail, yet no process can distinguish that run from another in which fewer than τ wake up; hence, no process knows that τ are awake.

1.4 Relation to Other Problems

We establish connections between the wakeup problem and two fundamental problems in distributed computing: the consensus problem and the leader election problem. These two problems lie at the core of many problems for fault-tolerant distributed applications [1, 7, 10, 13, 16, 20, 21, 32, 34, 43, 42, 48].

We show that: (1) any protocol that uses v values and solves the wakeup problem for $t < n/2$, $\tau > n/2$ and $p = 1$ can be transformed into t-resilient consensus and leader election protocols which use $8v$ values; and (2) any t-resilient consensus and leader election protocol that uses v values can be transformed into a t-resilient protocol which uses $4v$ values and solves the wakeup problem for any $\tau \leq \lfloor n/2 \rfloor + 1$ and $p = 1$.

Using the first result above, we can construct efficient solutions to both the consensus and leader election problems from solutions for the wakeup problem. The second result implies that the lower bound proved for the wakeup problem holds for these other two problems. As a consequence, the consensus and the leader election problems are space-equivalent in our model. This is particularly surprising since the two problems seem so different. The difficulty in leader election is breaking symmetry, whereas consensus is inherently symmetric.

2 Definitions and Notations

2.1 Protocols and Knowledge

An n-process *protocol* $P = (C, N, R)$ consists of a nonempty set C of *runs*, an n-tuple $N = (q_1, \ldots, q_n)$ of *process id's* (or *process*, for short), and an n-tuple $R = (R_1, \ldots, R_n)$ of sets of *registers*. Informally, R_i includes all the register that process q_i can access. We assume throughout this paper that $n \geq 2$.

A run is a pair of the form (f, S) where f is a function which assigns initial values to the registers in $R_1 \cup \ldots \cup R_n$ and S is a finite or infinite sequence of events. (When S is finite, we also say that the run is finite.) An *event* $e = (q_i, v, r, v')$ means that process q_i, in one atomic step, first reads a value v from register r and then writes a value v' into register r. We say that the event e *involves* process q_i and that process q_i performs a *test-and-set* operation on register r.

The set of runs is assumed to satisfy several properties; for example, it should be prefix closed. Because of lack of space, we do not give a complete list here but point out that these properties capture the fact that we are assuming that the processes are anonymous and identically programmed, are not synchronized, and that nothing can be assumed about the initial state of the shared memory.

The *value* of a register at a finite run is the last value that was written into that register, or its initial value if no process wrote into the register. A register r is said to be *local* if there exists an i such that $r \in R_i$ and for any $j \neq i, r \notin R_j$. A register is *shared* if it is not local. In this paper we restrict attention to protocols which have exactly one register which is shared by all the processes (i.e., $|R_1 \cap \ldots \cap R_n| = 1$) and all other registers are local. If S' is a prefix of S then the run (f, S') is *a prefix* of (f, S), and (f, S) is *an extension* of (f, S'). For any sequence S, let S_i be the subsequence of S containing all events in S which involve q_i.

Definition: Computations (f, S) and (f', S') are *equivalent with respect to* q_i, denoted by $(f, S) \overset{i}{\sim} (f', S')$, iff $S_i = S'_i$.

We are now ready to define the notion of knowledge in a shared memory environment. In the following, we use *predicate* to mean a set of runs.

Definition: For a process q_i, predicate b and finite run ρ, *process* q_i *knows* b *at* ρ iff for all ρ' such that $\rho \overset{i}{\sim} \rho'$, it is the case that $\rho' \in b$.

For simplicity, we assume that a process always takes a step whenever it is scheduled. A process that takes infinitely many steps in a run is said to be *correct* in that run; otherwise it is *faulty*. We say that an infinite run is *l-fair* iff at least l processes are correct in it.

2.2 Wakeup, Consensus and Leader Election Protocols

In this subsection we formally define the notions of t-resilient wakeup, consensus and leader election protocols ($0 \leq t \leq n$). We say that a process q_i is *awake* in a run if the run contains an event that involves q_i. The predicate "*at least* τ *processes are awake in run* ρ" is the set of all runs for which there exist τ different processes which are awake in the run. Note that a process that fails after taking a step is nevertheless considered to be awake in the run.

- A *wakeup protocol* with parameters n, t, τ and p is a protocol for n processes such that, for any $(n-t)$-fair run ρ, there exists a finite prefix of ρ in which at least p processes *know* that at least τ processes are awake in ρ.

  It is easy to see that a wakeup protocol exists only if $\max(p, \tau) \leq n - t$, and hence, from now on, we assume that this is always the case. We also assume that $\min(p, \tau) \geq 1$.

  In the following, whenever we speak about a solution to the wakeup problem without mentioning p, we are assuming that $p = 1$.

- A *t-resilient k-consensus protocol* is a protocol for n processes, where each process has a local read-only input register and a local write-once output

register. For any $(n-t)$-fair run there exists a finite prefix in which all the correct processes decide on some value from a set of size k (i.e., each correct process writes a *decision value* into its local output register), the decision values written by all processes are the same, and the decision value is equal to the input value of some process.

In the following, whenever we say "consensus" (without mentioning specific k) we mean "binary consensus", where the possible decision values are 0 and 1.

- A *t-resilient leader election protocol* is a protocol for n processes, where each process has a local write-once output register. For any $(n-t)$-fair run there exists a finite prefix in which all the correct processes decide on some value in $\{0,1\}$, and exactly one (correct or faulty) process decides on 1. That process is called the *leader*.

3 Fault-free solutions

In this section, we develop the *See-Saw protocol*, which solves the fault-free wakeup problem using a single 4-valued shared register. Then we show how the See-Saw protocol can be used to solve the k-valued consensus problem. Finally, we claim that it is impossible to solve the wakeup problem using only one shared bit.

To understand the See-Saw protocol, the reader should imagine a playground with a See-Saw in it. The processes will play the protocol on the See-Saw, adhering to strict rules. When each process enters the playground (wakes up), it sits on the up-side of the See-Saw causing it to swing to the ground. Only a process on the ground (or down-side) can get off and when it does the See-Saw must swing to the opposite orientation. These rules enforce a balance invariant which says that the number of processes on each side of the See-Saw differs by at most one (the heavier side always being down).

Each process enters the playground with two tokens. The protocol will force the processes on the bottom of the See-Saw to give away tokens to the processes on the top of the See-Saw. Thus, token flow will change direction depending on the orientation of the See-Saw. Tokens can be neither created nor destroyed. The idea of the protocol is to cause tokens to concentrate in the hands of a single process. A process seeing $2k$ tokens knows that at least k processes are awake. Hence, if it is guaranteed that eventually some process will see at least 2τ tokens, the protocol is by definition a wakeup protocol with parameter τ, even if the process does not know the value of τ and hence does not know when the goal has been achieved.

Following is the complete description of the See-Saw protocol. The 4-valued shared register is easily interpreted as two bits which we call the "token bit" and the "See-Saw" bit. The two states of the token bit are called "token present" and "no token present". We think of a public *token slot* which either contains a token or is empty, according to the value of the token bit. The two states of the See-Saw bit are called "left side down" and "right side down". The "See-Saw" bit describes a virtual See-Saw which has a left and a right side. The bit indicates which side is down (implying that the opposite side is up).

Each process remembers in private memory the number of tokens it currently possess and which of four states it is currently in with respect to the See-Saw: "never been on" "on left side", "on right side", and "got off". A process is said to be on the up-side of the See-Saw if it is currently "on left side" and the See-Saw bit is in state "right side down", or it is currently "on right side" and the See-Saw bit is in state "left side down". A process initially possesses two tokens and is in state "never been on".

We define the protocol by a list of rules. When a process is scheduled, it looks at the shared register and at its own internal state and carries out the first applicable rule, if any. If no rule is applicable, it takes a null step which leaves its internal state and the value in the shared register unchanged.

Rule 1: *(Start of protocol)* Applicable if the scheduled process is in state "never been on". The process gets on the up-side of the See-Saw and then flips the See-Saw bit. By "get on", we mean that the process changes its state to "on left side" or "on right side" according to whichever side is up. Since flipping the See-Saw bit causes that side to go down, the process ends up on the down-side of the See-Saw.

Rule 2: *(Emitter)* Applicable if the scheduled process is on the down-side of the See-Saw, has one or more tokens, and the token slot is empty. The process flips the token bit (to indicate that a token is present) and decrements by one the count of tokens it possesses. If its token count thereby becomes zero, the process flips the See-Saw bit and gets off the See-Saw by setting its state to "got off".

Rule 3: *(Absorber)* Applicable if the scheduled process is on the up-side of the See-Saw and a token is present in the token slot. The process flips the token bit (to indicate that a token is no longer present) and increments by one the count of tokens it possesses.

Note that if a scheduled process is on the down-side, has $2k-1$ tokens, and a token is present in the token slot, then, although no rule is applicable, the process nevertheless sees a total of $2k$ tokens and hence knows that k processes have waked up.

The two main ideas behind the protocol can be stated as invariants.

TOKEN INVARIANT: The number of tokens in the system is either $2n$ or $2n+1$ and does not change at any time during the protocol. (The number of tokens in the system is the total number of tokens possessed by all of the processes, plus 1 if a token is present in the token bit slot.)

BALANCE INVARIANT: The number of processes on the left and right sides of the See-Saw is either perfectly balanced or favors the down-side of the See-Saw by one process.

Theorem 3.1: *Let $t = 0$. The See-Saw protocol uses a 4-valued shared register and is a wakeup protocol for n, t, τ (and $p = 1$), where n and τ are arbitrary and $t = 0$. (Note that the rules for the protocol do not mention n or τ.)*

In applications of wakeup protocols, it is often desirable for the processes to know the value of τ so that a process learning that τ processes are awake can stop participating in the wakeup protocol and take some action based on that knowledge. The See-Saw protocol can be easily modified to have this property by adding a termination rule immediately after Rule 1:

Rule 1a: *(End of protocol)* Applicable if the scheduled process is on the See-Saw and sees at least 2τ tokens, where the number of tokens the process sees is the number it possesses, plus one if a token is present in the token slot. The process thus knows that τ processes have waked up. It gets off the See-Saw (i.e., terminates) by setting its state to "got off".

The See-Saw protocol can also be used to solve the leader election problem by electing the first process that sees $2n$ tokens. By adding a $5th$ value, everyone can be informed that the leader was elected, and the leader can know that everyone knows. Now, the leader can transmit an arbitrary message, for example a consensus value, to all the other processes without using any more new values through a kind of serial protocol. This leads to our next theorem.

Theorem 3.2: *In the absence of faults, it is possible to reach consensus on one of k values using a single 5-valued shared register.*

Finally, we claim that the See-Saw protocol cannot be improved to use only a single binary register. A slightly weaker result than Theorem 3.3 was also proved by Joe Halpern [27]. The question whether 3 values suffice is still open.

Theorem 3.3: *There does not exist a solution to the wakeup problem which uses only a single binary register when $\tau \geq 3$.*

4 Fault-tolerant solutions

In this section, we explore solutions to the wakeup problem which can tolerate $t > 0$ process failures.

The See-Saw protocol, presented in the previous section, cannot tolerate even a single crash failure for any $\tau > n/3$. The reason is that the faulty process may fail after accumulating $2n/3$ tokens, trapping two other processes on one side of the See-Saw, each with $2n/3$ tokens. When $\tau \leq n/3$, the See-Saw protocol can tolerate at least one failure. As the parameter τ decreases, the number of failures that the protocol can tolerate increases. This fact is captured in our first theorem.

Theorem 4.1: *The See-Saw protocol is a wakeup protocol for n, t, τ, where $\tau \leq ((2n-1)/(2t+1)+1)/2$.*

We note that the See-Saw protocol can tolerate up to $n/2 - 1$ *initial failures* [21, 49]. In the rest of this section, we present three t-resilient solutions to the wakeup problem. Notice that when the number of failures t is a constant, it is possible using a constant number of values for one process to learn that $n - t$ processes are awake.

Theorem 4.2: *For any $t < n/6$, there is a wakeup protocol which uses a single 8^{t+1}-valued register and works for any $\tau \leq n - t$.*

Theorem 4.3: *For any $t < n$, there is a wakeup protocol which uses a single n-valued register and works for any $\tau \leq n - t$.*

Theorem 4.4: *For any $t < n/2$, there is a wakeup protocol which uses a single $O(t)$-valued register and works for any $\tau \leq \lfloor n/2 \rfloor + 1$.*

5 A Lower Bound

In this section, we establish a lower bound on the number of shared memory values needed to solve the wakeup problem, where *only one* process is required to learn that τ processes are awake, assuming t processes may crash fail (thus $p = 1$). To simplify the presentation, we assume that $9 \leq t \leq 2n/3$ and $\tau > n/3$. Also, recall that we already assumed that $\tau \leq n - t$. For the rest of this section, let

$$W = \frac{t\sqrt{t} - t}{n - t}; \qquad U = \frac{t^2 - 1}{4(n - t)}; \tag{1}$$

$$\alpha = \frac{1}{\log_2(\frac{n-t}{t+2} + 3.5)}. \tag{2}$$

Note that $W \leq U$ since $t \geq 9$.

Theorem 5.1: *Let P be a wakeup protocol with parameters n, t and τ. Let V be the set of shared memory values used by P. Then $|V| \geq W^{\alpha}$.*

When we take t to be a constant fraction of n we get the following immediate corollary.

Corollary 5.1: *Let P be a wakeup protocol with parameters n, t and τ, where $t = n/c$. Let V be the set of shared memory values used by P and let $\gamma = 1/(2\log_2(c+2.5))$. Then, $|V| = \Omega(n^\gamma)$.*

Theorem 5.1 is immediate if V is infinite, so we assume throughout this section that V is finite. The proof consists of several parts. First we define a sequence of directed graphs whose nodes are shared memory values in V. Each component C of each graph in the sequence has a cardinality k_c and a weight w_c. We establish by induction that $k_c \geq \min(w_c, W)^\alpha$. Finally, we argue that in the last graph in the sequence, every component C has weight $w_c \geq W$. Hence, $|V| \geq k_c \geq W^\alpha$.

5.1 Reachability Graphs and Terminal Graphs

Let V be the alphabet of the shared register. We say that a value $a \in V$ appears m times in a given run if there are (at least) m different prefixes of that run where the value of the shared register is a.

$a \xrightarrow{u} b$ denotes that there exists a run in which *at most* u processes participate, the initial value of the shared register is a, and the value b appears at least once.

$a \stackrel{u}{\Longrightarrow} b$ denotes that there exists a run in which *exactly* u processes participate, each process that participates takes infinitely many steps, the initial value of the shared register is a, and the value b appears infinitely many times.

Clearly, $a \stackrel{r}{\Longrightarrow} b$ implies $a \xrightarrow{r} b$ but not vice versa. Also, for every a, there exists b such that $a \stackrel{r}{\Longrightarrow} b$.

We use the following graph-theoretic notions. A directed multigraph[1] G is *weakly connected* if the underlying undirected multigraph of G is connected. A multigraph $G'(V', E')$ is a *subgraph* of $G(V, E)$ if $E' \subseteq E$ and $V' \subseteq V$. A multigraph G' is a *component* of a multigraph G if it is a weakly connected subgraph of G and for any edge (a, b) in G, either both a and b are nodes of G' or both a and b are not in G'. A node is a *root* of a multigraph if there is a directed path from every other node in the multigraph to that node. A *rooted graph* (rooted component) is a graph (component) with at least one root. A *labeled* multigraph is a multigraph together with a label function that assigns a *weight* in $\mathbf{N}$ to each edge of G. The *weight* of a labeled multigraph is the sum of the weights of its edges.

We now define the notion of a reachability graph of a given protocol P.

[1] A multigraph can have several edges from a to b.

Definition: Let V be the set of shared memory values of protocol P. The *reachability graph G of protocol P* is the labeled directed multigraph with node set V which has an edge from node a to node b labeled with r iff $a \stackrel{r}{\Longrightarrow} b$ holds. (Note that there may be several edges with different labels between the same two nodes. Note also that G is finite since $a \stackrel{r}{\Longrightarrow} b$ implies that $r \leq |V|$.)

Definition: A graph C is *closed at node a w.r.t. G* if a is in C and for every node b in G, if (a, b) is an edge of G then b is in C.

Definition: A multigraph T is *terminal w.r.t. G* if T is a subgraph of G, all of T's components are rooted, and T has a component C with root a among its minimal weight components that is closed at node a w.r.t. G.

In the rest of the section we show that the reachability graph G of any wakeup protocol with parameters n, t, τ has size $\geq W^\alpha$. We do that by constructing a multigraph T which is *terminal w.r.t. G* and has size $\geq W^\alpha$. Theorem 5.1 follows from these facts.

5.2 Reachability Graphs

The reachability graphs are defined for all protocols. Now we concentrate on such graphs constructed from wakeup protocols only. We show that when the weight of a rooted component, say C, is sufficiently small, an edge exists with a label q from a root of C to a node not in C, and we can bound the size of q.

For later reference we call the following three inequalities,

$$\begin{array}{ll} \text{(i)} & pq + (p-1)w \leq n, \\ \text{(ii)} & pq \geq n - t, \\ \text{(iii)} & \max(q, w) < \tau \end{array}$$

the *zigzag inequalities*. These inequalities play an important role in our exposition.

Lemma 5.1: *Given reachability graph G of a wakeup protocol P with parameters n, t, τ and a rooted subgraph C of G with root a and weight w, if there exist positive integers p and q that satisfy the zigzag inequalities, then for any node b of G, if $a \stackrel{q}{\Longrightarrow} b$ is an edge of G then b is not in C.*

Proof: We assume to the contrary that there exists p and q that satisfy the zigzag inequalities, and there is an edge $a \stackrel{q}{\Longrightarrow} b$ such that b belongs to C. Let ρ be a q-fair run starting from a in which exactly q processes participate and b is written infinitely often. Since b is in C, there is a path from b to a such that the sum of all the labels of edges in that path is at most w and hence $b \xrightarrow{w} a$. This allows us to construct a run with pq non-faulty processes starting with a as follows:

> Run q processes according to ρ until b is written. Run w processes until a is written. (This

must be possible since $b \xrightarrow{w} a$.) Let these w processes fail. Run a second group of q processes according to ρ until b is written. Run a second group of w processes until a is written, and let them fail. Repeat the above until the p^{th} group of q processes have just been run and b has again been written. At this point, pq processes belong to still-active groups, and $(p-1)w$ processes have died. If any processes remain, let them die now without taking any steps. Now, an infinite run ρ' on the active processes can be constructed by continuing to run the first group according to ρ until b is written again, then doing the same for the second through p^{th} groups, and repeating this cycle forever.

The result is a pq-fair run. Moreover, no reliable process can distinguish this run from ρ, and hence no reliable process ever knows (in ρ') that more than q processes are awake. Also, obviously, no faulty process knows that more than w processes are awake. Since $\max(q,w) < \tau$ but at least $pq \geq n-t \geq \tau$ processes are awake in ρ', this leads to a contradiction to the assumption that P is a wakeup protocol. □

Lemma 5.2: *Assume $w \leq U$. Then the inequality*

$$x^2 - tx + w(n-t) \leq 0 \tag{3}$$

has a positive integer solution. The smallest positive integer solution for (3) is

$$q = \left\lceil \frac{t - |\sqrt{t^2 - 4w(n-t)}|}{2} \right\rceil \leq \frac{t}{2}. \tag{4}$$

There exists a positive integer p such that p and q satisfy the zigzag inequalities.

Proof: We first show that (3) has a positive solution. Using the quadratic formula, we get that the roots of (3) are

$$\frac{t - |\sqrt{t^2 - 4w(n-t)}|}{2} \quad \text{and} \quad \frac{t + |\sqrt{t^2 - 4w(n-t)}|}{2}.$$

Since $w \leq U$ the discriminant $t^2 - 4w(n-t) \geq 1$. Since the value of the discriminant is less than t^2 it follows that the roots are positive. Moreover, the difference of the two roots is at least 1; hence there is a positive *integer* x satisfying (3), and q is the least such integer. Moreover, since t is an integer, $t/2$ is either an integer or lies exactly half way between two integers, so inequality (4) holds.

Next we show that there exists a positive integer p such that p and q satisfy the inequalities (i) and (ii). Let $p = \lceil (n-t)/q \rceil$. The choice of p clearly satisfies (ii). Also from (3) it follows that

$$p = \left\lceil \frac{n-t}{q} \right\rceil \leq \frac{n-t}{q} + 1 \leq \frac{n+w}{q+w}$$

which implies (i).

Finally, we show that inequality (iii) is satisfied. Recall that we assume that $t \leq 2n/3$ and $\tau > n/3$. It follows from these assumptions that $\tau > t/2$. Since $q \leq t/2$, obviously $q < \tau$. Also, since $w \leq U$ and $t \leq 2n/3$, substituting in (1) gives $w \leq n/3$, and hence $w < \tau$. □

Lemma 5.3: *If $w \leq W$, then there exists positive integers p and q that satisfy the zigzag inequalities and*

$$q < \frac{w(n-t)}{t+2} + 3. \tag{5}$$

Proof: Recall that $W \leq U$, so in particular, $w \leq U$. Let q' be the smallest positive integer solution to (3). It follows from Lemma 5.2 that q' exists. Let q be the *smallest* positive integer for which there exists a positive integer p such that p and q satisfy the zigzag inequalities. It follows from Lemma 5.2 that q exists and $q \leq q'$.

If $q = 1$ than clearly $1 < w(n-t)/(t+2) + 2$ and the lemma holds. Assume $q > 1$. Since $q \leq q'$ it follows that

$$(q-1)^2 - t(q-1) + w(n-t) > 0.$$

Thus,

$$q < \frac{q^2 + 1 + t + w(n-t)}{t+2}. \tag{6}$$

By Lemma 5.2,

$$q \leq \left\lceil \frac{t - |\sqrt{t^2 - 4w(n-t)}|}{2} \right\rceil. \tag{7}$$

Since $w \leq W$, we can substitute W for w in inequality (7) and get that $q \leq \lceil \sqrt{t} \rceil < \sqrt{t} + 1$. Thus, $q^2 \leq t + 2\sqrt{t} + 1$, so it follows from (6) and the assumption that $t \geq 9$ that inequality (5) holds. □

5.3 Terminal Graphs

In this subsection, we show that the reachability graph G of any wakeup protocol with parameters n, t, τ, has at least one subgraph which is terminal w.r.t. G and has size $\geq W^{\alpha}$. We first prove that the weight of any rooted component of any terminal graph w.r.t. G has *weight* $\geq U$. Then we show that this implies that there exists a terminal graph w.r.t. G, all of whose rooted components have size $\geq W^{\alpha}$.

Lemma 5.4: *Let G be the reachability graph of a wakeup protocol with parameters n, t, τ and let T be terminal w.r.t. G. Any rooted component of T has weight $\geq U$.*

Proof: Assume to the contrary that T has a minimal-weight component C of weight $w < U$. Then, by Lemma 5.2, there exist positive integers q and p that satisfy the zigzag inequalities. From Lemma 5.1, there is a node b

not in C and an edge $a \stackrel{q}{\Longrightarrow} b$ in G. Therefore, T is not a terminal w.r.t. G, a contradiction. □

Lemma 5.5: *Let G be the reachability graph of a wakeup protocol with parameters n, t, τ. There exists a graph T which is terminal w.r.t. G, all of whose rooted components have size $\geq W^{\alpha}$.*

Proof: The following procedure constructs T by adding edges one at a time to an *initial* subgraph T_0 of G until step 2 fails. The *initial* subgraph T_0 consists of all the nodes of G. For each node a there is exactly one outgoing edge $a \stackrel{1}{\Longrightarrow} b$ in T_0. We note two facts about T_0: (1) for every edge $a \stackrel{1}{\Longrightarrow} b$, $a \neq b$, and (2) every component of T_0 has at least one root. Fact (1) follows from Lemma 5.1, choosing $q = 1$ and $p = n - t$ ($w = 0$); while (2) follows from the fact that the outdegree of any node is exactly one. Also, it follows from (1) that the weight and size of any component of T_0 is at least 2.

At any stage of the construction, every component of the graph built so far will have at least one root. Added edges always start at a root and end at a node of a different component. After adding an edge (a, b), the components of a and b are joined together into a single component whose root is the root of b's component, and the weight of the new component is the sum of the weights of the two original components plus the label of the edge from a to b.

Procedure for adding a new edge to T:

Step 1: *Select an arbitrary component C of minimal weight and an arbitrary root a of C.*

Step 2: *Find the smallest integer q for which there is an edge $a \stackrel{q}{\Longrightarrow} b$ in G such that b is not in C. This step fails if no such edge exists.*

Step 3: *Place the edge $a \stackrel{q}{\Longrightarrow} b$ into T.*

Let T_i be a graph that is constructed after i applications of the above procedure, where T_0 is an initial graph as defined above. Clearly, any such sequence $\{T_0, T_1, \ldots\}$ is finite and the last element is terminal w.r.t. G.

We prove by induction on i, the number of applications of the procedure, that for any graph T_i, all of the components of T_i are rooted, and for any rooted component C it is the case that $k \geq \min(w, W)^{\alpha}$, $k \geq 2$ and $w \geq 2$, where k is the size of C and w is its weight. This together with Lemma 5.4 and the fact that $W \leq U$ completes the proof.

Let $\beta = 1/\alpha$. As discussed before, each component C of T_0 has a root and has size k at least 2. The component C consists of exactly k edges with label 1, so its weight is also k. Hence, the base case holds since $\beta > 1$.

Since T_0 is a subgraph of T_i which also includes all nodes of T_i, it follows that the size and weight of any rooted component of T_i are both at least 2. Now, suppose the procedure adds an edge of label q from component C_1 of size k_1 and weight w_1 to component C_2 of size k_2 and weight w_2. By step 1, the new edge emanates from a minimal weight component, so $w_1 \leq w_2$. The weight w of the newly formed component is $w_1 + w_2 + q$, and the number of nodes k is $k_1 + k_2$. We show now that $k \geq \min(w, W)^{\alpha}$.

Clearly, if $w_2 \geq W$ then $\min(w_2, W)^{\alpha} = \min(w, W)^{\alpha}$ and $k_2 < k$, so by the induction hypothesis we are done. Hence, we assume that $w_2 < W$, so also $w_1 < W$. Since $w_1 < W$ it follows from Lemma 5.3 that there exist positive integers p' and q' that satisfy the zigzag inequalities and $q' < (w_1(n-t))/(t+2) + 3$; hence by Lemma 5.1 there is an edge of label q' from any root of C_1 to some node not in C_1. Thus, by the minimality of q (the weight of the edge in step 2), it follows that $q \leq q'$ which implies that $q < (w_1(n-t))/(t+2) + 3$; hence,

$$
\begin{aligned}
w &= w_1 + w_2 + q && (8)\\
&\leq \left(\frac{n-t}{t+2} + 1\right) w_1 + w_2 + 3 && (9)\\
&\leq \left(\frac{n-t}{t+2} + 2.5\right) w_1 + w_2. && (10)
\end{aligned}
$$

Let $k_1' = {w_1}^{\alpha}$, and $k_2' = {w_2}^{\alpha}$. Then $k_1' \leq k_2'$, $w_1 = {k_1'}^{\beta}$, and $w_2 = {k_2'}^{\beta}$. We claim that

$$
\begin{aligned}
&\left(\frac{n-t}{t+2} + 2.5\right) w_1 + w_2 \\
&\quad = \left(\frac{n-t}{t+2} + 2.5\right) {k_1'}^{\beta} + {k_2'}^{\beta} && (11)\\
&\quad \leq (k_1' + k_2')^{\beta}. && (12)
\end{aligned}
$$

It is not difficult to see that since $(n-t)/(t+2) + 3.5 = 2^{\beta}$, equality holds for $k_1' = k_2'$. As k_2' is increased to be larger than k_1', the right side increases more rapidly than the left side since $\beta > 1$; hence, the inequality holds. Finally, by the induction hypothesis, $k_1 \geq {w_1}^{\alpha} = k_1'$ and $k_2 \geq {w_2}^{\alpha} = k_2'$. Hence,

$$(k_1' + k_2')^{\beta} \leq (k_1 + k_2)^{\beta} = k^{\beta}. \qquad (13)$$

Putting equations (8)–(13) together gives $w \leq k^{\beta}$, so $k \geq w^{\alpha} \geq \min(w, W)^{\alpha}$. □

Theorem 5.1 follows immediately from Lemma 5.5.

6 Relation to Other Problems

In this section we show that there are efficient reductions between the wakeup problem for $\tau = \lfloor n/2 \rfloor + 1$ and the consensus and leader election problems. Hence, the wakeup problem can be viewed as a fundamental

problem that captures the inherent difficulty of these two problems. The following Lemma shows that in order to decide on some value in a t-resilient consensus protocol, it is always necessary (and in some cases also sufficient) to learn first that at least $t+1$ processes have waked up, and similarly in order to be elected in a t-resilient leader election protocol, it is always necessary to learn that at least $t+1$ processes have waked up. An immediate consequence of the lemma is that there is no consensus or leader election protocol that can tolerate $\lceil n/2 \rceil$ failures.

Lemma 6.1: *(1) Any t-resilient consensus (leader election) protocol is a t-resilient wakeup protocol for any $\tau \leq t+1$ and $p = n - t$ (p=1); (2) For any $t < n/3$, there exists t-resilient consensus and leader election protocols which are not t-resilient wakeup protocols for any $\tau \geq t+2$.*

Theorem 6.1: *Any protocol that solves the wakeup problem for any $t < n/2$, $\tau > n/2$ and $p = 1$, using a single v-valued shared register, can be transformed into a t-resilient consensus (leader election) protocol which uses a single $8v$-valued ($4v$-valued) shared register.*

From Theorems 6.1 and 4.4, it follows that for any $t < n/2$, there is a t-resilient consensus (leader election) protocol that uses an $O(t)$-valued shared register. Next we show that the converse of Theorem 6.1 also holds. That is, the existence of a t-resilient consensus or leader election protocol which uses a single v-valued shared register implies the existence of a t-resilient wakeup protocol for any $\tau \leq \lfloor n/2 \rfloor + 1$ which uses a single $O(v)$-valued shared register.

Theorem 6.2: *Any t-resilient protocol that solves the consensus or leader election problem using a single v-valued shared register can be transformed into a t-resilient protocol that solves the wakeup problem for any $\tau \leq \lfloor n/2 \rfloor + 1$ which uses a single $4v$-valued shared register.*

It follows from Theorem 6.2 that the lower bound we proved in Section 5 for the wakeup problem when $\tau = \lfloor n/2 \rfloor + 1$ also applies to the consensus and leader election problems. Finally, an immediate corollary of Theorem 6.1 and Theorem 6.2 is that the consensus and leader election problems are space-equivalent. That is, there is a t-resilient consensus protocol that uses an $O(t)$-valued shared register iff there is a t-resilient leader election protocol that uses an $O(t)$-valued shared register.

7 Conclusions

We study the fundamental new wakeup problem in a new model where all processes are programmed alike, there is no global synchronization, and it is not possible to simultaneously reset all parts of the system to a known initial state.

Our results are interesting for several reasons:

- They give a quantitative measure of the cost of fault-tolerance in shared memory parallel machines in terms of communication bandwidth.
- They apply to a model which more accurately reflects reality.
- They relate recent results from three different active research areas in parallel and distributed computing, namely:
  - Results in shared memory systems [2, 11, 19, 31, 36, 38, 39, 46, 50, 51].
  - The theory of knowledge in distributed systems [6, 14, 15, 17, 18, 22, 28, 25, 29, 30, 26, 33, 37, 40, 41].
  - Self stabilizing protocols [3, 4, 8, 9, 12, 24, 35, 47].
- They give a new point of view and enable a deeper understanding of some classical problems and results in cooperative computing.
- They are proved using techniques that will likely have application to other problems in distributed computing.

Acknowledgement

We thank Joe Halpern for helpful discussions.

References

[1] K. Abrahamson. On achieving consensus using shared memory. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 291–302, 1988.

[2] B. Bloom. Constructing two-writer atomic registers. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 249–259, 1987.

[3] G. M. Brown, M. G. Gouda, and C.-L. Wu. Token systems that self-stabilize. *IEEE Trans. on Computers*, 38(6):845–852, June 1989.

[4] J. E. Burns and J. Pachl. Uniform self-stablilizing rings. *ACM Trans. on Programming Languages and Systems*, 11(2):330–344, 1989.

[5] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, April 1989.

[6] M. Chandy and J. Misra. How processes learn. *Journal of Distributed Computing*, 1:40–52, 1986.

[7] E. Chang and R. Roberts. An improved algorithm for decentralized extrema-finding in circular configuration of processes. *Communications of the ACM*, 22(5):281–283, 1979.

[8] E. W. Dijkstra. Self-stablizing systems in spite of distributed control. *Communications of the ACM*, 17:643–644, 1974.

[9] E. W. Dijkstra. A belated proof of self-stabilization. *Journal of Distributed Computing*, 1:5–6, 1986.

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

[11] D. Dolev, E. Gafni, and N. Shavit. Toward a non-atomic era: *l*-exclusion as a test case. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 78–92, 1988.

[12] S. Dolev, A. Israeli, and S. Moran. Self stabilization of dynamic systems assuming only read write atomicity. submitted for publication, 1989.

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

[14] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment i: Crash failures. In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 149–169. Morgan Kaufmann, 1986.

[15] R. Fagin, Y. J. Halpern, and M. Vardi. A model theoretic analysis of knowledge. In *Proc. 25th IEEE Symp. on Foundations of Computer Science*, pages 268–278, 1984.

[16] M. J. Fischer. The consensus problem in unreliable distributed systems (a brief survey). In M. Karpinsky, editor, *Foundations of Computation Theory*, pages 127–140. Lecture Notes in Computer Science, vol. 158, Springer-Verlag, 1983.

[17] M. J. Fischer and N. Immerman. Foundations of knowledge for distributed systems. In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 171–185. Morgan Kaufmann, March 1986.

[18] M. J. Fischer and N. Immerman. Interpreting logics of knowledge in propositional dynamic logic with converse. *Information Processing Letters*, 25(3):175–181, May 1987.

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

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

[21] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[22] M. J. Fischer and L. D. Zuck. Reasoning about uncertainty in fault-tolerant distributed systems. In M. Joseph, editor, *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 142–158. Lecture Notes in Computer Science, vol. 331, Springer-Verlag, 1988.

[23] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir. The NYU ultracomputer—designing an MIMD parallel computer. *IEEE Trans. on Computers*, pages 175–189, February 1984.

[24] M. G. Gouda. The stabilizing philosopher: Asymmetry by memory and by action. *Science of Computer Programming*, 1989. To appear.

[25] V. Hadzilacos. A knowledge theoretic analysis of atomic commitment protocols. In *Proc. 6th ACM Symp. on Principles of Database Systems*, pages 129–134, 1987.

[26] J. Halpern and L. Zuck. A little knowledge goes a long way: Simple knowledge-based derivations and correctness proofs for a family of protocols. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 269–280, August 1987.

[27] Y. J. Halpern. personal communication.

[28] Y. J. Halpern. Reasoning about knowledge: An overview. In *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the 1986 Conference*, pages 1–17. Morgan Kaufmann, 1986.

[29] Y. J. Halpern and R. Fagin. A formal model of knowledge, action, and communication in distributed systems: Preliminary report. In *Proc. 4th ACM Symp. on Principles of Distributed Computing*, pages 224–236, 1985.

[30] Y. J. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. In *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, pages 50–61, 1984.

[31] P. M. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 276–290, 1988.

[32] D. S. Hirschberg and J.B. Sinclair. Decentralized extrema-finding in circular configuration of processes. *Communications of the ACM*, 23:627–628, 1980.

[33] S. Katz and G. Taubenfeld. What processes know: Definitions and proof methods. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pages 249–262, August 1986.

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

[35] H. S. M. Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 2:91–95, 1979.

[36] L. Lamport. The mutual exclusion problem: Statement and solutions. *Journal of the ACM*, 33:327–348, 1986.

[37] D. Lehmann. Knowledge, common knowledge and related puzzles. In *Proc. 3rd ACM Symp. on Principles of Distributed Computing*, pages 62–67, 1984.

[38] C. M. Loui and H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing Research*, 4:163–183, 1988.

[39] N. A. Lynch and M. J. Fischer. A technique for decomposing algorithms which use a single shared variable. *Journal of Computer and System Sciences*, 27(3):350–377, December 1983.

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

[41] R. Parikh and R. Ramanujam. Distributed processes and the logic of knowledge. In R. Parikh, editor, *Proceedings of the Workshop on Logic of Programs*, pages 256–268, 1985.

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

[43] G. L. Peterson. An $O(n \log n)$ unidirectional algorithm for the circular extrema problem. *ACM Trans. on Programming Languages and Systems*, 4(4):758–762, 1982.

[44] G. H. Pfister and et. al. The IBM research parallel processor prototype (RP3): Introduction and architecture. In *Proceedings International Conference on Parallel Processing*, 1985.

[45] A. G. Ranade, S. N. Bhatt, and S. L. Johnsson. The fluent abstract machine. Technical Report YALEU/DCS/TR–573, Department of Computer Science, Yale University, January 1988.

[46] G. Taubenfeld. Leader election in the presence of $n-1$ initial failures. *Information Processing Letters*, 33:25–28, 1989.

[47] G. Taubenfeld. Self-stabilizing Petri nets. Technical Report YALEU/DCS/TR–707, Department of Computer Science, Yale University, May 1989.

[48] G. Taubenfeld, S. Katz, and S. Moran. Impossibility results in the presence of multiple faulty processes. In *Proc. of the 9th FCT-TCS conference, Bangalore, India*, December 1989. Previous version appeared as Technion TR-#492, January 1988.

[49] G. Taubenfeld, S. Katz, and S. Moran. Initial failures in distributed computations. *International Journal of Parallel Programming*, 1990. To appear. Previous version appeared as Technion TR-#517, August 1988.

[50] G. Taubenfeld and S. Moran. Possibility and impossibility results in a shared memory environment. In J.C. Bermond and M. Raynal, editors, *Proc. 3rd International Workshop on Distributed Algorithms*, pages 254–267. Lecture Notes in Computer Science, vol. 392, Springer-Verlag, 1989.

[51] P. M. B. Vitanyi and B. Awerbuch. Atomic shared register access by asynchronous hardware. In *Proc. 27th IEEE Symp. on Foundations of Computer Science*, pages 223–243, 1986.