Computation in Networks of Passively Mobile Finite-State Sensors*

Dana Angluin
angluin-dana@cs.yale.edu

James Aspnes
aspnes-james@cs.yale.edu

Zoë Diamadi
diamadi-zoe@cs.yale.edu

Michael J. Fischer
fischer-michael@cs.yale.edu

René Peralta
peralta@cs.yale.edu

Department of Computer Science
Yale University
New Haven, CT 06520–8285

ABSTRACT

We explore the computational power of networks of small resource-limited mobile agents. We define two new models of computation based on pairwise interactions of finite-state agents in populations of finite but unbounded size. With a fairness condition on interactions, we define the concept of stable computation of a function or predicate, and give protocols that stably compute functions in a class including Boolean combinations of threshold-k, parity, majority, and simple arithmetic. We prove that all stably computable predicates are in NL. With uniform random sampling of pairs to interact, we define the model of conjugating automata and show that any counter machine with $O(1)$ counters of capacity $O(n)$ can be simulated with high probability by a protocol in a population of size n. We prove that all predicates computable with high probability in this model are in $P \cap RL$. Several open problems and promising future directions are discussed.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*distributed networks, network communications, network topology, wireless communication*; F.1.1 [**Computation by Abstract Devices**]: Models of Computation; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*parallelism and concurrency, probabilistic computation*; E.1 [**Data**]: Data Structures—*distributed data structures*

General Terms

Theory, Algorithms, Performance

Keywords

Diffuse computation, finite-state agent, intermittent communication, mobile agent, sensor net, stable computation

1. SCENARIO: A FLOCK OF BIRDS

Suppose we have equipped each bird in a particular flock with a sensor that can determine whether the bird's temperature is elevated or not, and we wish to know whether at least 5 birds in the flock have elevated temperatures. We assume that the sensors are quite limited: each sensor has a constant number of bits of memory and can respond to a global start signal, and two sensors can communicate only when they are sufficiently close to each other.

In this scenario, the sensors are mobile, but have no control over how they move, that is, they are **passively mobile**. Initially, we assume that the underlying pattern of movement guarantees a fairness condition on the interactions: every pair of birds in the flock repeatedly come sufficiently close to each other for their sensors to communicate.

Under these assumptions, there is a simple protocol ensuring that every sensor eventually contains the correct answer. At the global start signal, each sensor makes a measurement, resulting in a 1 (elevated temperature) or 0 (not elevated temperature) in a counter that can hold values from 0 to 4. When two sensors communicate, one of them sets its counter to the sum of the two counters, and the other one sets its counter to 0. If two counters ever sum to at least 5, the sensors go into a special alert state, which is then copied by every sensor that encounters them. The output of a sensor is 0 if it is not in the alert state, and 1 if it is in the alert state. If we wait a sufficient interval after we issue the global

*Supported in part by NSF grants CCR-9820888, CCR-0098078, CSE-0081823, CNS-0305258, and by ONR grant N00014-01-1-0795.

PODC'04, July 25–28, 2004, St. John's, Newfoundland, Canada.
Copyright 2004 ACM 1-58113-802-4/04/0007 ...$5.00.

start signal, we can retrieve the correct answer from any of the sensors.

Now consider the question of whether at least 5% of the birds in the flock have elevated temperatures. Is there a protocol to answer this question in the same sense, without assumptions about the size of the flock? In Section 3, we show that such a protocol exists. More generally, we are interested in fundamental questions about the computational power of this and related models of interactions among members of a distributed population of finite-state agents.

2. A WIDER VIEW

Most work in distributed algorithms assumes that agents in a system are computationally powerful, capable of storing non-trivial amounts of data and carrying out complex calculations. But in systems consisting of massive amounts of cheap, bulk-produced hardware, or of small mobile agents that are tightly constrained by the systems they run on, the resources available at each agent may be severely limited. Such limitations are not crippling if the system designer has fine control over the interactions between agents; even finite-state agents can be regimented into cellular automata with computational power equivalent to linear space Turing machines. But if the system designer cannot control these interactions, it is not clear what the computational limits are.

Sensor networks are a prime example of this phenomenon. Each sensing unit is a self-contained physical package including its own power supply, processor and memory, wireless communication capability, and one or more sensors capable of recording information about the local environment of the unit. Constraints on cost and size translate into severe limitations on power, storage, processing, and communication. Sensing units are designed to be deployed in large groups, using local low-power wireless communication between units to transmit information from the sensors back to a base station or central monitoring site.

Research in sensor networks has begun to explore the possibilities for using distributed computation capabilities of networks of sensors in novel ways to reduce communication costs. Aggregation operations, such as count, sum, average, extrema, median, or histogram, may be performed on the sensor data in the network as it is being relayed to the base station [12, 13]. Flexible groups of sensors associated with targets in spatial target tracking can conserve resources in inactive portions of the tracking area [7, 17]. Though sensors are usually assumed to be stationary or nearly so, permitting strategies based on relatively stable routing, this assumption is not universal in the sensor-network literature. For example, an assumption of random mobility and packet relay dramatically increases the throughput possible for communication between source-destination pairs in a wireless network [9].

The flock of birds scenario illustrates the question of characterizing what computations are possible in a cooperative network of passively mobile finite-state sensors. The assumptions we make about the motion of the sensors are that it is passive (not under the control of the sensors), sufficiently rapid and unpredictable for stable routing strategies to be infeasible, and that each pair of sensors will repeatedly be close enough to communicate using a low-power wireless signal.

There is a global start signal transmitted by the base station to all the sensors simultaneously to initiate a computation. When they receive the global start signal, the sensors take a reading (one of a finite number of possible input values) and attempt to compute some function or predicate of all the sensor values. This provides a "snapshot" of the sensor values, rather than the continuous stream of sensor values more commonly considered. Sensors communicate in pairs and do not have unique identifiers; thus, they update their states based strictly on the pair of their current states and on the role each plays in the interaction—one acting as initiator and the other as responder.

In Section 3, we define a model of computation by pairwise interactions in a population of identical finite-state agents. Assuming a fairness condition on interactions, we define the concept of stable computation of a function or predicate by a population protocol. In Section 4, we consider the question of what predicates can be stably computed when interactions can occur between all pairs of agents. We give protocols for threshold-k, parity, majority, and simple arithmetic functions, as well as closure results that allow us to define a useful expression language that captures a subset of the power of this model. We also show that every predicate computable in this model is in nondeterministic log space. An open problem is to give an exact characterization of the computational power of stable computation in this model. In Section 5, we show that the all-pairs case is the weakest for stably computing predicates by showing that it can be simulated by any population that cannot be separated into non-interacting subpopulations. The questions of what additional predicates can be computed for reasonable restrictions on the interactions and what properties of the underlying interaction graph can be stably computed by a population are open.

In Section 6, we obtain the model of conjugating automata by adding a uniform sampling condition on interactions to the assumption that interactions are enabled between all pairs of agents. This allows us to consider computations that are correct with high probability and to address questions of expected resource use. We show that this model has sufficient power to simulate, with high probability, a counter machine with $O(1)$ counters of capacity $O(n)$. We further show that Boolean predicates computable with high probability in this model are in $P \cap RL$. This gives a partial characterization of the set of predicates computable by such machines, but finding an exact characterization is still open. In Section 7, we describe other related work, and in Section 8 we discuss some of the many intriguing questions raised by these models.

3. A FORMAL MODEL

We define a model that generalizes the flock of birds scenario from Section 1.

3.1 Population Protocols

A **population protocol** $\mathcal{A}$ consists of finite **input and output alphabets** X and Y, a finite set of **states** Q, an **input function** $I : X \to Q$ mapping inputs to states, an **output function** $O : Q \to Y$ mapping states to outputs, and a **transition function** $\delta : Q \times Q \to Q \times Q$ on pairs of states. If $\delta(p, q) = (p', q')$, we call $(p, q) \mapsto (p', q')$ a **transition**, and we define $\delta_1(p, q) = p'$ and $\delta_2(p, q) = q'$.

As a simple illustration, we formalize a version of the count-to-five protocol from Section 1. The six states are $q_0, \ldots, q_5$. The input and output alphabets are $X = Y =$

$\{0, 1\}$. The input function I maps 0 to q_0 and 1 to q_1. The output function O maps all states except q_5 to 0 and the state q_5 to 1. The transition function $\delta(q_i, q_j)$ is defined as follows: if $i + j \geq 5$, then the result is (q_5, q_5); if $i + j < 5$ then the result is (q_{i+j}, q_0).

A population protocol runs in a population of any finite size n. A **population** $\mathcal{P}$ consists of a set A of n **agents** together with an irreflexive relation $E \subseteq A \times A$ that we interpret as the directed edges of an **interaction graph**. E describes which agents may interact during the computation. Intuitively, an edge $(u, v) \in E$ means that u and v are able to interact, with u playing the role of **initiator** and v playing the role of **responder** in the interaction. Note that the distinct roles of the two agents in an interaction is a fundamental assumption of asymmetry in our model; symmetry-breaking therefore does not arise as a problem within the model. Though most of the present paper concerns the case in which E consists of all ordered pairs of distinct elements from A, we give definitions appropriate for general E.

A **population configuration** is a mapping $C : A \rightarrow Q$ specifying the state of each member of the population. Let C and C' be population configurations, and let u, v be distinct agents. We say that C goes to C' via **encounter** $e = (u, v)$, denoted $C \xrightarrow{e} C'$, if

$$\begin{aligned} C'(u) &= \delta_1(C(u), C(v)) \\ C'(v) &= \delta_2(C(u), C(v)) \\ C'(w) &= C(w) \text{ for all } w \in A - \{u, v\}. \end{aligned}$$

We say that C can go to C' in one step, denoted $C \rightarrow C'$, if $C \xrightarrow{e} C'$ for some encounter $e \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_k = C'$, such that $C_i \rightarrow C_{i+1}$ for all i, $0 \leq i < k$, in which case we say that C' is **reachable** from C.

A **computation** is a finite or infinite sequence of population configurations $C_0, C_1, C_2, \ldots$ such that for each i, $C_i \rightarrow C_{i+1}$. An infinite computation is **fair** if for every pair of population configurations C and C' such that $C \rightarrow C'$, if C occurs infinitely often in the computation, then C' also occurs infinitely often in the computation.

3.2 Computation by Population Protocols

As with nondeterministic Turing machines, we define a notion of input and output, and we capture the input-output behavior of population protocols by relations. Unlike Turing machines, population protocols do not halt, so there is no obvious fixed time at which to view the output of the population. Rather, we say that the output of the computation stabilizes if it reaches a point after which no agent can subsequently change its output value, no matter how the computation proceeds thereafter. Stability is a global property of the population configuration, so individual agents in general do not know when stability has been reached. However, with suitable stochastic assumptions on the rate at which interactions occur, it is possible to bound the expected number of interactions until the output stabilizes. We explore this approach in Section 6.

An **input assignment** is a function $x : A \rightarrow X$ and describes the inputs to a population protocol. We let $\mathcal{X} = X^A$ denote the set of all input assignments. The inputs are represented by an **input configuration** C_x, where $C_x(w) = I(x(w))$ for all $w \in A$. We naturally extend I to a mapping from input assignments to configurations by writing $I(x) = C_x$. In words, if x assigns input symbol σ to agent u, then agent u's state in configuration $I(x)$ is $I(\sigma)$.

An **output assignment** is a function $y : A \rightarrow Y$ and describes the outputs of a population protocol. We let $\mathcal{Y} = Y^A$ denote the set of all output assignments. Given a configuration C, we let y_C denote the corresponding output assignment, where $y_C(w) = O(C(w))$ for all $w \in A$. We naturally extend O to a mapping from configurations to output assignments by writing $O(C) = y_C$. In words, if agent u is in state q in configuration C, then agent u's output symbol in output assignment $O(C)$ is $O(q)$.

A configuration C is said to be **output-stable** if $O(C') = O(C)$ for all C' reachable from C. Note that we do not require that $C = C'$, only that their outputs be equal. An infinite computation **output-stabilizes** if it contains an output-stable configuration C, in which case we say that it **stabilizes to output** $y = O(C)$. It is immediate that an infinite computation stabilizes to at most one output.

The output of a finite computation is the output of its last configuration. The output of an infinite computation that stabilizes to output y is y; the output is undefined if the computation does not stabilize to any output. Because of the nondeterminism inherent in the choice of encounters, the same initial configuration may lead to different computations that stabilize to different outputs.

A population protocol $\mathcal{A}$ running in a population $\mathcal{P}$ **stably computes an input-output relation** $R_{\mathcal{A}}$ as follows. For each $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, $R_{\mathcal{A}}(x, y)$ holds if and only if there is a fair computation of $\mathcal{A}$ beginning in configuration $I(x)$ that stabilizes to output y. In the special case that $R_{\mathcal{A}}$ is single-valued[1], we write $F_{\mathcal{A}}(x) = y$ for $R_{\mathcal{A}}(x, y)$ and say that $\mathcal{A}$ **stably computes the partial function** $F_{\mathcal{A}} : \mathcal{X} \rightarrow \mathcal{Y}$.

Continuing our count-to-five illustration, assume that the agents are $u_1, \ldots, u_6$ and the interaction graph is complete.[2] Let the input assignment x be described by the vector

$$(0, 1, 0, 1, 1, 1),$$

assigning input symbols to the agents $u_1 \ldots, u_6$ in that order. The corresponding input configuration is

$$I(x) = (q_0, q_1, q_0, q_1, q_1, q_1),$$

which leads to the following possible computation:

$$\begin{aligned} (q_0, q_1, q_0, q_1, q_1, q_1) &\xrightarrow{(2,4)} (q_0, q_2, q_0, q_0, q_1, q_1) \\ &\xrightarrow{(6,5)} (q_0, q_2, q_0, q_0, q_0, q_2) \\ &\xrightarrow{(2,6)} (q_0, q_4, q_0, q_0, q_0, q_0) \\ &\xrightarrow{(3,2)} (q_0, q_0, q_4, q_0, q_0, q_0). \end{aligned}$$

The configurations reachable from the last one above are those with five agents assigned q_0 and one agent assigned q_4, and the outputs of all of them are equal to

$$(0, 0, 0, 0, 0, 0).$$

Therefore, $R((0, 1, 0, 1, 1, 1), (0, 0, 0, 0, 0, 0))$ holds, where R is the input-output relation computed by this protocol. In fact, R is singled-valued, so we can write

$$F(0, 1, 0, 1, 1, 1) = (0, 0, 0, 0, 0, 0).$$

[1] A relation R is single-valued if $\forall x \forall y \forall z (R(x, y) \wedge R(x, z) \Rightarrow y = z)$.

[2] An intersection graph is complete if (u, v) is a directed edge for all agents u, v with $u \neq v$.

In this example, we could have designed our protocol so that the configurations themselves stopped changing, but this illustrates the fact that we only require the outputs to stop changing. In the next section, we show that the irrelevance of agent identities to this computation is a general phenomenon.

3.3 Functions on Other Domains

As defined in Section 3.2, population protocols compute partial functions from $\mathcal{X}$ to $\mathcal{Y}$. In order to use population protocols to compute functions on other domains, we need suitable input and output encoding conventions.

Functions with Multiple Arguments.

Let f be a function over domain X^n. The **agent-based input convention** assumes an ordered set of n agents and gives the i^{th} argument to the i^{th} agent, $1 \leq i \leq n$. Depending on the interaction graph and output conventions, the particular ordering chosen for the agents might or might not affect the function computed.

Predicates on $\mathcal{X}$.

A predicate on $\mathcal{X}$ can be regarded as a function from $\mathcal{X}$ to $\{0, 1\}$. The **predicate output convention** assumes $Y = \{0, 1\}$ and requires *every* agent to agree on the output. A population protocol $\mathcal{A}$ **stably computes a predicate** if and only if its input-output relation is single-valued and for all input assignments x, if the predicate is true of x, then $F_{\mathcal{A}}(x)$ is the output assignment that maps every agent to 1; otherwise, $F_{\mathcal{A}}(x)$ is the output assignment that maps every agent to 0. The formal count-to-five protocol described above stably computes the predicate of x that is true if and only if x assigns 1 to at least 5 different agents. This generalizes easily to functions from $\mathcal{X}$ to the set of output symbols Y; we require the final outputs of all the agents to be equal to the correct value of the function.

Integer Functions.

Let $f : \mathbb{Z}^k \to \mathbb{Z}^\ell$ be a partial function on integer vectors. Integer input and output values are represented diffusely across the population rather than being stored locally by individual agents. We describe an encoding convention that can represent $O(1)$ integers with absolute values bounded by $O(n)$ in a population protocol with n agents.

Let Γ be a set. (We will generally take Γ to be either X, Y, or Q.) A **k-place integer encoding convention over** Γ is a mapping $\rho : \Gamma \to \mathbb{Z}^k$. Thus, ρ associates a k-vector of integers to each element of Γ. We extend ρ to a mapping $\Gamma^A \to \mathbb{Z}^k$ by summing over all agents: for $\gamma \in \Gamma^A$, we define

$$\rho(\gamma) = \sum_{u \in A} \rho(\gamma(u)),$$

that is, $\rho \circ \gamma$ maps each agent to a k-vector, and we sum over all agents to obtain the vector $\rho(\gamma)$ that is **represented by** γ.

An integer input k-vector is encoded by a k-place encoding convention ρ^X over the input set X. An integer output ℓ-vector is encoded by an ℓ-place encoding convention ρ^Y over the output set Y. During internal computation, $\mathcal{A}$ can maintain an m-vector encoded by ρ^Q over the state set Q.

Using these encoding conventions, we say that $\mathcal{A}$ **stably computes** f if the following conditions hold:

1. For every $r \in \mathbb{Z}^k$ in the domain of f, there exists $x \in \mathcal{X}$ such that $\rho^X(x) = r$.
2. For every $x \in \mathcal{X}$, if $\rho^X(x)$ is in the domain of f then there exists $y \in \mathcal{Y}$ such that $R_{\mathcal{A}}(x, y)$ holds.
3. For every $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, if $R_{\mathcal{A}}(x, y)$ holds then $\rho^Y(y) = f(\rho^X(x))$.

Note that $\mathcal{A}$ can stably compute integer function f even though $R_{\mathcal{A}}$ is not single valued. It might happen, for example, that $r = \rho^X(x)$, $R_{\mathcal{A}}(x, y_1)$ and $R_{\mathcal{A}}(x, y_2)$ for $y_1 \neq y_2$, but $\rho^Y(y_1) = \rho^Y(y_2) = f(r)$.

Example of an Integer Function.

We describe a population protocol to compute the function $f(m) = \lfloor m/3 \rfloor$, the integer quotient of m and 3. We take $X = Y = \{0, 1\}$ and let the input and output encoding functions be the identity. Thus, an input assignment x represents m if the number of agents assigned 1 is m, and similarly for output assignments.

The states are ordered pairs (i, j) of integers such that $0 \leq i \leq 2$ and $0 \leq j \leq 1$, and the state encoding function is also the identity. The input map I maps 1 to the state $(1, 0)$ and 0 to the state $(0, 0)$. The output map O maps state (i, j) to j.

The transition function is defined as follows: if $i + k \leq 2$ and $i, j > 0$ then $\delta((i, 0), (k, 0)) = ((i + k, 0), (0, 0))$, and if $i + k > 2$ then $\delta((i + k - 3, 0), (0, 1))$. All other transitions are defined to leave the pair of states unchanged. Transitions of the first type can accumulate two 1's to a 2, but leave the sum of the first coordinates of the states unchanged. Transitions of the second type reduce the sum of the first coordinates of the states by 3 and increase the sum of the second coordinates by 1. Eventually, no more transitions of either type will be possible, and the sum of the second coordinates will be the integer quotient of m and 3, as desired. (If the output map were changed to the identity, this protocol would compute the ordered pair consisting of the remainder of m modulo 3 and the integer quotient of m and 3.)

4. ALL-PAIR INTERACTIONS

In this section, we restrict attention to populations in which the interaction graph is complete. Under this assumption, stably computable input-output relations are invariant under renaming of the agents, that is, if π is a permutation on A and $R_{\mathcal{A}}(x, y)$, then $R_{\mathcal{A}}(x \circ \pi, y \circ \pi)$. In the case of predicates, the output assignment y is a constant function, which implies that $y \circ \pi = y$ and the output is determined by the multiset of input symbols. Therefore the stably computable predicates are symmetric.

Because population protocols depend only on the states of agents, not on their names, we define a standard agent set $A_n = \{1, \ldots, n\}$ of size n, and we let $\mathcal{P}_n$ be a population of size n consisting of the complete interaction graph on A_n.

Let $\{f_n\}$ be a family of Boolean functions such that $f_n : \{0, 1\}^n \to \{0, 1\}$ for all $n \geq 1$. A population protocol $\mathcal{A}$ **stably computes** the family $\{f_n\}$ if for every $n \geq 1$, when $\mathcal{A}$ is run on population $\mathcal{P}_n$, it stably computes the function f_n, with the agent-based input convention (agent i gets the i-th argument) and the predicate output convention (eventually every agent agrees on the correct output). By the discussion above, every stably computable family of Boolean functions is symmetric.

More generally, we consider languages, that is, predicates on X^*, the set of all finite strings of input symbols. We say that population protocol $\mathcal{A}$ stably computes L if for every $n \geq 1$, when $\mathcal{A}$ is run on population $\mathcal{P}_n$, it stably computes the characteristic function of L restricted to strings of length n, with the agent-based input and predicate output conventions. If L is stably computable, it contains every permutation of each of its elements.

4.1 Stably Computable Predicates

We begin by considering what families of Boolean functions are stably computable.

Closure Properties.

Let $\{f_n\}$ and $\{g_n\}$ be families of Boolean functions stably computable by population protocols $\mathcal{A}$ and $\mathcal{B}$. By complementing the output map of $\mathcal{A}$, the family of negations $\{f'_n\}$ is stably computable. By complementing the input map of $\mathcal{A}$, the family of functions $\{h_n\}$, where h_n is equal to f_n with its inputs complemented, is stably computable. Product constructions with $\mathcal{A}$ and $\mathcal{B}$ yield population protocols that stably compute the conjunction or disjunction of the outputs of $\{f_n\}$ and $\{g_n\}$. Formally, we have:

Lemma 1. *Let $\{f_n\}$ and $\{g_n\}$ be families of Boolean functions. If $\{f_n\}$ and $\{g_n\}$ are stably computable, then so are $\{\neg f_n\}$, $\{f_n \wedge g_n\}$, and $\{f_n \vee g_n\}$.*

Thus, in addition to predicates such as "at least 5 ones" we can compute predicates such as "fewer than 5 ones", "at least 5 zeros", "at most 5 zeros", "exactly 5 ones", "exactly 5, 7 or 9 ones" and so on. Though it is not immediately obvious, parity and majority are also stably computable by population protocols.

Parity.

The value of parity is 1 if there are an odd number of 1's in the input, and 0 if there are an even number of 1's in the input. Note that computing the parity function with the predicate output convention is different from computing the remainder modulo 2 in the integer output encoding convention, because in the predicate output convention eventually all the agents must have the correct output symbol.

Our construction for parity uses a state consisting of two bits: the data bit and the live bit. Initially, the data bit is equal to the input bit, and the live bit is 1. For each state, the output bit is equal to the data bit. When two agents meet whose live bits are both 1, one sets its live bit to 0, and the other sets its data bit to the mod 2 sum of their data bits. When an agent with live bit 0 meets an agent with live bit 1, the former copies the data bit of the latter.

In this protocol, the mod 2 sum of the product of the live bit and the data bit over all the agents in the population is invariant and equal to the mod 2 sum of the inputs. Eventually, exactly one agent has its live bit set to 1. At that point, its data bit is the correct output. Once there is a single live bit set to 1, eventually every other agent copies the (correct) data bit from that agent.

The live bit ensures a tree-structured computation aggregating the data bits, with the final result at the root of the tree. This generalizes to the computation of the product of all the input values in a commutative semigroup. Thus, all the symmetric regular languages are stably computable, e.g., deciding whether the number of 1's is i modulo m for constants i and m.

Majority.

The value of the majority function is 1 if there are more 1's than 0's in the input; otherwise, it is 0.

The states of our protocol consist of a live bit and a counter with values in the set $\{-1, 0, 1\}$. Initially, the live bit is 1, and the counter is -1 if the input is 0 and 1 if the input is 1. The output is 1 if the counter is 1; otherwise, it is 0. When two agents with live bit equal to 1 meet, if the sum of their counters is in the set $\{-1, 0, 1\}$, then both set their counters to the sum, and one of the agents sets its live bit to 0; otherwise, they do nothing. When an agent with live bit equal to 0 meets an agent with live bit equal to 1, the former copies the counter value of the latter.

In this protocol, the sum of the counters for all agents with live bit equal to 1 is invariant and equal to the number of 1's minus the number of 0's in the input. Eventually there will remain one or more live bits equal to 1, with no more combinations possible, which means that the associated counter(s) have a single value -1, 0, or 1, indicating that the number of 1's in the input is less than, equal to, or greater than the number of 0's in the input, respectively. After this point, every other agent will copy the common counter value, and their outputs will also be correct.

A generalization with counters capable of holding integer values between $-k$ and k inclusive determines whether at least a fraction $1/(k+1)$ of the inputs are 1's, demonstrating the existence of a population protocol to detect whether at least 5% of the flock of birds have elevated temperatures, as claimed in Section 1.

Arithmetic Functions.

Recall from Section 3.3 the distributed representation of $O(1)$ integers of absolute value $O(n)$ and the example of the protocol to divide by 3. Using similar encodings and ideas, there are population protocols to compute the sum of two integers, the product of an integer and a constant, the integer quotient of an integer and a constant, the value of an integer modulo a constant, and a constant function (equal to k for all inputs). Given stable inputs, these protocols output-stabilize and thus can be composed freely.

A Stably Computable Expression Language.

Putting together the base functions and closure results, we can define an expression language such that the expressible predicates are stably computable by population protocols. This gives a lower bound on the set of stably computable predicates: whether every stably computable predicate is so expressible is an open problem.

Expressions are defined as follows. For each input symbol $\sigma \in X$, there is a variable N_σ representing the number of agents assigned the symbol σ by the input assignment. Constants are nonnegative integers. Each term is a constant, a variable, or the sum of two terms, or the product of a constant and a term, or the integer quotient of a term and a nonzero constant, or the remainder of a term modulo a nonzero constant. Atomic expressions are formed from two terms and one of the predicates: $=$, $\leq$, $<$, $\geq$, $>$. An expression is either an atomic expression or the negation of an expression, the conjunction of two expressions, or the disjunction of two expressions. Each expression is either true or false of a given input assignment, by the usual semantics.

For example, the count-to-five problem is expressible as $(N_1 \geq 5)$, the parity problem as $((N_1 \bmod 2) = 1)$, and the majority problem as $(N_1 > N_0)$. The question of whether the number of 1's is between 15% and 20% of the total population can be expressed as

$$((17N_1 \geq 3N_0) \wedge (4N_1 \leq N_0)).$$

This idea extends to predicates on non-binary input alphabets, so that if $X = \{a, b, c\}$, we can express the predicate that the number of a's, b's, and c's are equal by the expression $(N_a = N_b) \wedge (N_b = N_c)$.

THEOREM 2. *Any predicate expressed in the language described above is stably computable.*

(As already noted, whether the converse holds is an open problem.)

PROOF SKETCH. Because the set of stably computable predicates is closed under complementation, union, and intersection, it suffices to show that the terms and atomic expressions of the language described above are stably computable.

Given a term of the language, we use the distributed representation of integers and allocate a component of the state for each subterm of the expression, including variables and constants. The input map places the value of N_σ into the component allocated to it. For each other subterm of the expression, there is a subprotocol (operating in parallel with all the others) to compute its value in the correct component of the state, generally as a function of the values being computed in other components. These use the protocols for sum, multiplication by a constant, division by a constant, and remainder modulo a nonzero constant, and constant functions, which each eventually stabilize to the correct outputs.

For atomic expressions, it suffices to consider $(t_1 \geq t_2)$, again because the stably computable predicates are closed under negation, intersection and union. We use the above method of computing the values of all the subterms of this expression, and run (also in parallel) a protocol to compare the values of t_1 and t_2 and propagate the results to all other agents. This protocol is similar to the majority protocol described above. □

We note in passing that there is an exact relationship between our expression language and the semilinear sets. Fix an ordering $\sigma_1, \ldots, \sigma_k$ of the input alphabet, and define the Parikh map Ψ from X^* to $\mathbb{N}^k$ by

$$\Psi(x) = (\#_{\sigma_1}(x), \ldots, \#_{\sigma_k}(x)),$$

where $\#_{\sigma_i}(x)$ is the number of occurrences of σ in x. Using Ginsburg and Spanier's characterization of the semilinear sets as those that can be defined in Presburger arithmetic [8], it is straightforward (but beyond the scope of this paper) to show that the semilinear sets in $\mathbb{N}^k$ are precisely the images under Ψ of predicates expressible in the language defined above. Thus, we may rephrase one of our open problems: are there stably computable predicates whose corresponding subsets of $\mathbb{N}^k$ are not semilinear?

4.2 Predicates Not Stably Computable

Theorem 2 gives a partial characterization of the stably computable predicates in the population model with all pairs enabled. We do not know if this characterization is complete. However, we can obtain an upper bound on the set of predicates stably computable in this model by showing that it is contained in the complexity class *NL*.

Because stably computable predicates in this model are symmetric, it is sufficient to represent a population configuration by the multiset of states assigned to the agents. Since there are $|Q|$ possible states and the population consists of n agents, each population configuration can thus be represented by $|Q|$ counters of $\lceil \log n \rceil$ bits each. A population protocol step can be simulated by drawing two elements of the multiset, applying the transition function and returning the resulting two elements to the multiset.

If there is a population protocol $\mathcal{A}$ that stably computes a language $L \subseteq X^*$, then there is a nondeterministic Turing machine to accept L in space $O(\log n)$. To accept input x, the Turing machine must verify two conditions: that there is a configuration C reachable from $I(x)$ in which all states have output 1, and there is no configuration C' reachable from C in which some state has output 0. The first condition is verified by guessing and checking a polynomial-length sequence of multiset representations of population configurations reaching such a C. The second condition is the complement of a similar reachability condition. It is in nondeterministic $O(\log n)$ space because this class is closed under complement [11]. It follows that:

THEOREM 3. *All predicates stably computable in the model with all pairs enabled are in the class NL.*

It is an open problem to characterize exactly the power of this model of stable computation. Concretely, we conjecture that predicates such as "the number of 1's is a power of 2" and "the number of c's is the product of the number of a's and the number of b's" are not stably computable by population protocols. Our intuition is that the model lacks the ability to sequence or iterate computations, and we suspect that a pumping lemma of some form exists for the model.

5. RESTRICTED INTERACTIONS

Some interaction graphs may permit very powerful computations by population protocols; for example, a population whose interaction graph is a directed line can easily simulate a linear-space Turing machine. In this section, we prove that the complete interaction graph we have been assuming up until now is in a sense the *weakest* structure for stably computing predicates, in that any weakly-connected interaction graph can simulate it.

THEOREM 4. *For any population protocol $\mathcal{A}$, there exists a population protocol $\mathcal{A}'$ such that for every n, if $\mathcal{A}$ stably computes predicate P on the standard population $\mathcal{P}_n$, and if $\mathcal{P}'$ is any population with agents 1,2,...,n and a weakly-connected interaction graph, then $\mathcal{A}'$ stably computes P on $\mathcal{P}'$.*

PROOF SKETCH. The key idea is to have any interaction in $\mathcal{A}'$ choose nondeterministically between swapping the states of the two interacting agents–which eventually brings any two simulated agents together–or simulating an interaction in $\mathcal{A}$; most of the details of the simulation involve implementing this nondeterministic choice with deterministic transitions. To do so, the state space in $\mathcal{A}'$ is augmented to add two "batons", S (for the initiator) and R (for responder), which move somewhat independently of the simulated agents. The presence or not of the two batons is used to control what effect an interaction has: an interaction that involves no batons swaps the states; an interaction that involves one baton moves the baton; and an interaction that involves both batons simulates a transition in $\mathcal{A}$. This mechanism assumes that n is at least 4 to give room to move the batons out of the way; smaller n values are handled by a separate protocol running in parallel that overrides the output of the main protocol if it detects $n \le 3$.

Let $Q' = Q \times \{D, S, R, -\}$, where Q is the state space of $\mathcal{A}$, D is a default initial state of the baton field, S marks the initiator baton, R marks the responder baton, and $-$ marks a "blank" or absent baton. Writing ordered pairs in Q' using simple concatenation, e.g., qD for (q, D), the transition function δ' is given by

$$\begin{array}{rcl}
(xD, yD) & \mapsto & (xS, yR) \\
(xD, y*) & \mapsto & (x-, y*) \\
(x*, yD) & \mapsto & (x*, y-) \\
\\
(xS, yS) & \mapsto & (xS, y-) \\
(xR, yR) & \mapsto & (xR, y-) \\
\\
(xS, y-) & \leftrightarrow & (x-, yS) \\
(xR, y-) & \leftrightarrow & (x-, yR) \\
\\
(x-, y-) & \leftrightarrow & (y-, x-) \\
\\
(xS, yR) & \mapsto & (x'R, y'S) \\
(yR, xS) & \mapsto & (y'S, x'R)
\end{array}$$

where x and y range over all states in Q, $*$ represents any non-D baton, and $(x', y') = \delta(x, y)$.

The first group of transitions consumes all initial D batons, producing at least one S and at least one R baton; the second group eventually reduces the set of non-blank batons to exactly one S and one R. The remaining groups implement (a) baton movement, (b) state swapping, and (c) $\mathcal{A}$-transitions. Note that $\mathcal{A}$-transitions also swap batons; this is done to allow S and R batons to pass each other in narrow graphs, which may be necessary to bring duplicates together in the initial stage.

The simulated $\mathcal{A}$ execution is obtained by ignoring both the batons and agent order. Most of the proof of correctness involves showing that fairness holds in the simulation, that is, that any state that is reachable infinitely often is reached infinitely often. This is done by showing that any $\mathcal{A}$-transition can be simulated by a finite sequence of $\mathcal{A}'$-transitions, involving moving the relevant states to adjacent agents using state swaps (which may require additional transitions to move the batons off each edge before the states on its endpoints can be swapped), moving the S and R batons onto the adjacent agents, and triggering an $\mathcal{A}$-transition. □

6. RANDOMIZED INTERACTIONS: CONJUGATING AUTOMATA

"Stability" is probably not a strong enough guarantee for most practical situations, but it is the best we can offer given only the fairness condition. To make stronger guarantees, we must put some constraints on the interactions between members of the population.

Let us add a probabilistic assumption on how the next pair to interact is chosen. Many assumptions would be reasonable to study. We consider one of the simplest: the ordered pair to interact is chosen at random, independently and uniformly from all ordered pairs corresponding to edges in the interaction graph. When the interaction graph is complete, this is the model of **conjugating automata**, inspired by models introduced by Diamadi and Fischer to study the acquisition and propagation of knowledge about trustworthiness in populations of interacting agents [4].

Random pairing is sufficient to guarantee fairness with probability 1, so any protocol that stably computes a predicate g in a fair model computes g with probability 1 on every input in the corresponding random-pairing model, assuming both run on the same population.

However, probabilities also allow us to consider problems where we only compute the correct answer with high probability, or to describe the expected number of interactions until a protocol converges. Given a function f mapping $\mathcal{X}$ to $\mathcal{Y}$, a population protocol $\mathcal{A}$, and an input x, we define the probability that $\mathcal{A}$ computes f on input x to be the probability of all computations beginning with $I(x)$ that stabilize with output $f(x)$.

For example, for the parity protocol, the expected number of interactions in a computation until there is just one live bit equal to 1 is $\Theta(n^2)$, and the expected number of further interactions until every other member of the population has interacted with the unique member with live bit equal to 1 is $\Theta(n^2 \log n)$. Thus the expected total number of interactions until the output is correct is $\Theta(n^2 \log n)$. In general, we are interested in protocols that accomplish their tasks in an expected number of interactions polynomial in n, the population size.[3]

Generalizing this argument, we obtain the following by structural induction:

THEOREM 5. *Let P be a predicate defined by the language of Section 4.1. Then there is a randomized population protocol that computes P with probability 1, where the population converges to the correct answer in expected total number of interactions $O(k_P n^2 \log n)$, where k_P is a constant depending on P.*

6.1 The Benefits of a Leader

Simulating Counters.

If we are allowed to designate a leader in the input configuration, that is, one agent that starts in a distinguished state, then the leader can organize the rest of the population to simulate a counter machine with $O(1)$ counters of capacity $O(n)$, with high probability. We assume throughout this section that the interaction graph is complete.

We use the representation described in Section 3.3 for integers in arithmetic computations. For a simulation of k counters in which counter i can take on a maximum value of $c_i n$, each state is mapped to a k-tuple of nonnegative integers in $[0, c_1] \times \cdots \times [0, c_k]$. The sum of component i over the population gives the current contents of counter i. We assume that the inputs to the counter machine are supplied in designated counters and the leader simulates the finite-state control of the counter machine.

[3] Note that such protocols do not terminate with a final answer; they remain capable of resuming indefinitely.

To decrement counter i, the leader waits to encounter an agent with component i of its state greater than zero, and decrements it. Incrementing counter i is similar; component i must be less than its maximum value c_i. These operations will happen with probability 1, assuming that they are possible. However, testing counter i for zero is different; the leader must attempt to decide whether there are any agents with component i greater than zero. We give a method that is correct with high probability. It is the ability to make (possibly incorrect) decisions that enables effective sequencing and iteration of computations in this model.

The leader initially labels one other agent (the timer) with a special mark. The leader waits for one of two events: (1) an interaction with an agent with a nonzero in component i, or (2) k consecutive interactions with the timer. If an event of type (1) occurs first, then the simulated counter is certainly not zero. The event (2) has low probability, so if it occurs first, the probability is high that the leader has encountered every other agent in the meantime and may (with a small probability of error) conclude that the value of simulated counter i is zero. The parameter k controls the probability of error, at the expense of increasing the expected number of interactions.

Lemma 6. *If the leader marks one other agent as a timer, the expected total number of interactions until the leader encounters the timer k times in a row is $\Theta(n^{k+1})$.*

Proof Sketch. Let $T(k)$ be the number of encounters involving the leader until the leader encounters the timer k times in a row. Then $T(k) = T(k-1)+1+\frac{n-1}{n}\cdot T(k)$ because the leader must first encounter the timer $k-1$ times in a row. Once that happens, another agent is encountered by the leader. With probability $\frac{n-1}{n}$, it is not the timer, in which case we must start all over again. Solving the recurrence relation for $T(k)$, we get $T(k) = n \cdot (T(k-1)+1)$, which expands to $T(k) = \sum_{i=1}^{k} n^i \in \Theta(n^k)$. Since the probability that the leader is involved in a given encounter is $2/n$, the result follows. □

Rather than sum error bounds for individual counter operations, we analyze the error associated with the macro-operations on counters:

Lemma 7. *For sufficiently large k, the operations of zeroing a counter or zeroing a counter while adding its contents to one or more other counters can be performed in an expected total number of interactions $\Theta(n^{k+1})$ with $O((1/n)^{k-1} \log n)$ probability of error. The same holds for the operations of multiplying by a constant, integer quotient with a constant, or remainder modulo a constant.*

Proof Sketch. We show the bounds on the number of interactions and probability of error for zeroing counter i, which is the fundamental operation. The leader marks one other agent as a timer and waits for one of the following two events: (1) an encounter with an agent with a non-zero component i, in which case, the agent's component i is set to zero, (2) k consecutive encounters with the timer, in which case the leader concludes that the zeroing operation is complete. The bound on the expected number of interactions follows directly from Lemma 6.

To calculate the probability of error, note that a failure occurs only if the leader encounters the timer k times in a row before it encounters every agent with a nonzero component i. Let f_i be the probability that the leader encounters the timer $k-i$ times in a row given that m members of the population have a nonzero component i. We want to compute f_0. We do so by solving the recurrence

$$f_i = f_{i+1} \cdot \frac{1}{n} + f_0 \cdot \frac{n-m-1}{n},$$

eventually deriving

$$f_0 = (1-\frac{1}{n})(\frac{1}{n})^k/(1-\frac{1}{n}-\frac{n-m-1}{n}+\frac{n-m-1}{n}\cdot(\frac{1}{n})^k),$$

which can be simplified to show that $f_0 < (\frac{1}{n})^{k-1}/m$. The probability of error is bounded above by $\sum_{m=1}^{n}(\frac{1}{n})^{k-1}/m$, which is in $O((1/n)^{k-1} \cdot \log n)$.

To achieve the other operations, we elaborate the zeroing operation as follows: (1) increment each of a collection of other counters for each decrement of the source counter, to copy the source value to these destinations, (2) increment another counter c times for each decrement of the source counter, to multiply the source counter by c, (3) increment another counter once for each c decrements of the source counter, to divide the source counter by c and simultaneously find its value modulo c. The analysis of interaction and error bounds is similar. □

How to Elect a Leader.

If we do not have a unique leader in the input configuration, it is possible to establish one using the ideas of the live bit, as in the parity and majority protocols of Section 4.1, and the timer mark of Section 6.1.

At the global start signal, every agent receives its input symbol (which it remembers for the duration of the computation), sets its live bit equal to 1, and clears its timer mark (indicating that it is not a timer). Any agent whose live bit equals 1 begins an initialization phase: it marks the first non-timer agent that it encounters as a timer and attempts to initialize every other agent. It uses the event of encountering a timer k times in a row to determine the end of the initialization phase.

Of course, at first every agent is attempting to run the initialization phase, so there will be general chaos. Whenever two agents with live bit equal to 1 encounter each other, one (the loser) sets its live bit to 0, and the other (the winner) keeps its live bit 1. If the loser has already marked a timer, the winner waits until it encounters a timer and turns it back into a non-timer before proceeding. The winner then restarts the initialization phase (not creating another timer if it has already released one). When initialized, agents with live bit equal to 0 revert to a state representing only their input and their live bit, but they retain their timer status.

If an agent with live bit equal to 1 completes the initialization phase, it begins the computation (e.g., simulating a counter machine, as in the preceding section). If during the computation it encounters another agent with live bit equal to 1, the two proceed as indicated above, one setting its live bit to 0, and the other restarting the initialization phase, with appropriate housekeeping to ensure retrieval of the extra timer, if any.

After a period of unrest lasting an expected $\Theta(n^2)$ interactions, there will be just one agent with live bit equal to 1. After the interaction eliminating the last rival, this lucky winner will succeed in initializing all other agents with high probability (because there is only one timer in the population) and proceed with the computation as the unique leader. If and when the counter machine halts, the unique leader can propagate that fact (along with the output, if a function of one output is being computed) to all the other agents. If there have been no errors during the (final) simulation, the output of every configuration in the rest of the computation is correct.

We have just shown how to carry out the operations of a counter machine with high probability. Using a standard construction due to Minsky [15], we can now simulate randomized log-space Turing machines with high probability.

Corollary 8. *Let $f(x)$ be a function in randomized log-space, where the input x is represented in unary. Then for any fixed c, there is a population protocol that, with n members, computes $f(x)$ for any $x \leq n$ with probability of error $O(n^{-c})$ in expected time polynomial in n.*

6.2 Simulating Randomized Population Protocols

In this section, we show either deterministic polynomial time or randomized logarithmic space is sufficient to recognize predicates computable with probability at least $1/2 + \epsilon$ by a population protocol with random pairing.

Suppose that a population protocol $\mathcal{A}$ computes a predicate P with probability at least $1/2 + \epsilon$. Then P can be computed by a polynomial-time Turing machine. As before, we assume that a string x of symbols from X represents an input assignment x to $\mathcal{A}$, so that n represents both the input length and the population size.

On input x, a polynomial-time Turing machine can construct the matrix representing the Markov chain whose states are the multiset representations of the population configurations reachable from $I(x)$, since there are at most $n^{|Q|}$ of them. Solving for the stationary distribution of the states, the Turing machine can determine a set of configurations of probability greater than 1/2 that all have the same output (which must be correct.) The Turing machine then writes this common output to its output tape and halts.

Also under these assumptions, P can be computed by a randomized Turing machine with probability $1/2 + \epsilon'$ using space $O(\log n)$. A randomized Turing machine simulates the population protocol by using a finite number of $O(\log n)$-bit counters to keep track of the number of members of the population in each state. Using coin flips, it simulates drawing a random pair of population members and updating the counters according to the transition function of $\mathcal{A}$. By running the simulation for long enough, the randomized Turing machine can be almost certain of being in a terminal strongly connected component of the states of the Markov chain, at which point the Turing machine halts and writes the output of the current configuration on its output tape.

To wait sufficiently long, the randomized Turing machine allocates a counter of $c\lceil \log n \rceil$ bits and flips a coin before each simulated interaction, adding 1 to the counter on heads, and clearing the counter on tails. The simulation is stopped when the counter overflows, that is, when there have been at least n^c consecutive heads. This gives an expected number of simulated interactions before termination of at least 2^{n^c}.

We have just shown:

Theorem 9. *The set of predicates accepted by a randomized population protocol with probability $1/2 + \epsilon$ is contained in $P \cap RL$.*

7. OTHER RELATED WORK

In a Petri net, a finite collection of tokens may occupy one of a finite set of places, and transition rules specify how the tokens may move from place to place. [4] Viewing the states of a population protocol as places, and the population members as tokens, our models can also be interpreted as particular kinds of Petri nets. Randomized Petri nets were introduced by Volzer [16] using a transition rule that does not depend on the number of tokens in each input place, as ours does in the case of conjugating automata.

The Chemical Abstract Machine of Berry and Boudol [2] is an abstract machine designed to model a situation in which components move about a system and communicate when they come into contact, based on a metaphor of molecules in a solution governed by reaction rules. A concept of enforced locality using membranes to confine subsolutions allows the machines to implement classical process calculi or concurrent generalizations of the lambda calculus.

Ibarra, Dang, and Egecioglu [10] consider a related model of catalytic P systems. They show that purely catalytic systems with one catalyst define precisely the semilinear sets, and also explore other models equivalent in power to vector addition systems. The relationships between these models and ours is an intriguing topic.

Brand and Zafiropulo [3] define a model of communicating processes consisting of a collection of finite state machines that can communicate via pre-defined FIFO message queues, and focus on general properties of protocols defined in the model, such as the possibility of deadlock or loss of synchronization.

Milner's bigraphical reactive systems [14] address the issues of modeling locality and connectivity of agents by two distinct graph structures. In this work the primary focus is upon the expressiveness of the models, whereas we consider issues of computational power and resource usage.

8. DISCUSSION AND OPEN PROBLEMS

In addition to the open problem of characterizing the power of stable computation, many other intriguing questions and directions are suggested by this work. One direction we have explored [1] is to define a novel storage device, the **urn**, which contains a multiset of tokens from a finite alphabet and functions as auxiliary storage for a finite control with input and output tapes, analogous to the pushdown or work tape of traditional models. Access to the tokens in the urn is by uniform random sampling, making it similar to the model of conjugating automata.

We have primarily considered the case of a complete interaction graph, which we have shown in Theorem 9 provides the least computational power of all weakly-connected interaction graphs in the stable computation model. The question of characterizing the power of stable computations on particular restricted interaction graphs remains open. We can also consider the interaction graph itself as part of the input and ask what interesting properties of its underlying graph can be stably computed by a population protocol. This problem may have applications in analyzing the structure of deployed sensor networks.

An interesting restriction of our model is to consider only **one-way communication** between the two agents in an interaction, that is, the transition function δ can be restricted

[4] See [5, 6] for surveys of Petri nets.

to change only the state of the responder in the interaction, keeping the state of the initiator the same. Although there are still protocols to decide whether the number of 1's in the input is at least k, this condition appears to restrict the stably computable predicates severely.

The models in this paper assume a "snapshot" of the inputs is taken when the global start signal is received. A model accommodating streaming inputs, as is typically assumed in sensor networks, would be very interesting.

We have assumed uniform sampling of pairs to interact, but for some applications it may make sense to consider other sampling rules. One idea is weighted sampling, in which population members are sampled according to their weights, possibly depending on their current states. We conjecture that with reasonable restrictions on the weights, weighted sampling yields the same power as uniform sampling. Other sampling rules might be based on more accurate models of patterns of interaction in populations of interest.

The interaction rules we consider are deterministic and specify pairwise interactions. What happens if the rules are nondeterministic, or specify interactions of larger groups, or allow the interaction to increase or decrease the population?

We give bounds on the expected total number of interactions, but other resource measures may be more appropriate in some applications. For many applications, interactions happen in parallel, so that the total number of interactions may not be well correlated with wall-clock time; defining a useful notion of time is a challenge. Alternatively, if we consider only the number of interactions in which at least one state changes (which might be correlated with the energy required by the computation), then the bounds can be finite even in the stable computation model, and the expected bounds can be smaller in the conjugating automata model.

9. ACKNOWLEDGMENTS

The authors wish to thank Richard Yang for valuable advice regarding these ideas, David Eisenstat for the parity construction and other discussions, and the anonymous reviewers for their thoughtful comments and suggestions.

10. REFERENCES

[1] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta. Urn automata. Technical Report YALEU/DCS/TR–1280, Yale University Department of Computer Science, Nov. 2003.

[2] G. Berry and G. Boudol. The Chemical Abstract Machine. *Theoretical Computer Science*, 96:217–248, 1992.

[3] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, Apr. 1983.

[4] Z. Diamadi and M. J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1–2):72–82, Mar. 2001. Also appears as Yale Technical Report TR–1207, January 2001, available at URL `ftp://ftp.cs.yale.edu/pub/TR/tr1207.ps`.

[5] J. Esparza. Decidability and complexity of Petri net problems-an introduction. In G. Rozenberg and W. Reisig, editors, *Lectures on Petri Nets I: Basic models.*, pages 374–428. Springer Verlag, 1998. Published as LNCS 1491.

[6] J. Esparza and M. Nielsen. Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994.

[7] Q. Fang, F. Zhao, and L. Guibas. Lightweight sensing and communication protocols for target enumeration and aggregation. In *Proceedings of the 4th ACM International Symposium on Mobile ad hoc networking & computing*, pages 165–176. ACM Press, 2003.

[8] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas, and languages. *Pacific Journal of Mathematics*, 16:285–296, 1966.

[9] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486, 2002.

[10] O. H. Ibarra, Z. Dang, and O. Egecioglu. Catalytic p systems, semilinear sets, and vector addition systems. *Theor. Comput. Sci.*, 312(2-3):379–399, 2004.

[11] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, Oct. 1988.

[12] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000.

[13] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a Tiny AGgregation service for ad-hoc sensor networks. In OSDI 2002: Fifth Symposium on Operating Systems Design and Implementation, December, 2002.

[14] R. Milner. Bigraphical reactive systems: basic theory. Technical report, University of Cambridge, 2001. UCAM-CL-TR-523.

[15] M. L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall Series in Automatic Computation. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1967.

[16] H. Volzer. Randomized non-sequential processes. In *Proceedings of CONCUR 2001-Concurrency Theory*, pages 184–201, Aug. 2001.

[17] F. Zhao, J. Liu, J. Liu, L. Guibas, and J. Reich. Collaborative signal and information processing: An information directed approach. *Proceedings of the IEEE*, 91(8):1199–1209, 2003.