Terminating Population Protocols via Some Minimal Global Knowledge Assumptions*

Othon Michail[1,2], Ioannis Chatzigiannakis[1,2], and Paul G. Spirakis[1,2]

[1] Computer Technology Institute & Press "Diophantus" (CTI), Patras, Greece
[2] Computer Engineering and Informatics Department (CEID), University of Patras
{michailo,ichatz,spirakis}@cti.gr

Abstract. We extend the population protocol model with a *cover-time service* that informs a walking state every time it covers the whole network. This is simply a known upper bound on the cover time of a random walk. This allows us to introduce *termination* into population protocols, a capability that is crucial for any distributed system. By reduction to an oracle-model we arrive at a very satisfactory lower bound on the computational power of the model: we prove that it is *at least as strong as a Turing Machine of space* $\log n$ *with input commutativity*, where n is the number of nodes in the network. We also give a $\log n$-space, but nondeterministic this time, upper bound. Finally, we prove interesting similarities of this model to linear bounded automata.

1 Introduction

Networks of tiny artifacts will play a fundamental role in the computational environments and applications of tomorrow. As a result, over the last decade, there has been a strong focus on theoretical models of pervasive systems, consisting of great numbers of computationally restricted, communicating entities. One such model, called the *Population Protocol* (*PP*) model, has been recently introduced by Angluin *et al.* [AAD$^+$06]. Their aim was to model sensor networks consisting of tiny computational devices (called *agents*) with sensing capabilities that follow some unpredictable and uncontrollable mobility pattern. Due to the minimalistic nature of their model, the class of computable predicates was proven [AAER07] to be fairly small: it is the class of *semilinear predicates* [GS66], which does not support e.g. multiplications, exponentiations, and many other important operations on input variables. Additionally, population protocols do not halt. No

* This work has in part been supported by the EU (European Social Fund - ESF) and Greek national funds through (i) the Operational Programme "Education and Lifelong Learning" (EdLL), under the title "Foundations of Dynamic Distributed Computing Systems" (FOCUS), and (ii) the National Strategic Reference Framework (NSRF) (Regional Operational Programme - Western Greece) under the title "Advanced Systems and Services over Wireless and Mobile Networks" (number 312179).

A.W. Richa and C. Scheideler (Eds.): SSS 2012, LNCS 7596, pp. 77–89, 2012.
© Springer-Verlag Berlin Heidelberg 2012

agent of the population running the protocol can know whether the computation is completed. The agents forever interact in pairs while their outputs (but not necessarily their states) stabilize to a certain value.

An interesting question that quickly emerged was whether complex computations could be performed by using simple protocols and combining their functionality. Given the protocols stabilizing behavior, their sequential execution was impossible. To circumvent this problem, Angluin *et al.* introduced the *stabilizing inputs PPs* [AAC+05] and they showed that multiple protocols can run in parallel and once one stabilized the others could run correctly (by taking appropriate actions to restore correct execution) using the stabilized output of the former as their input. This approach is, however, fairly slow in terms of the number of interactions (provided some probabilistic assumption on the interaction pattern) since it requires to implement phase clocks based on epidemic protocols (see [AAE08]).

In this work, we follow an alternative approach. We augment the original model of computation with a *cover-time service* (we abbreviate the new model as *CTS*) that informs a walking state every time it covers the whole network. This is simply a known upper bound on the cover time of a random walk. This allows us to introduce *termination* into population protocols, a capability that is crucial for any distributed system. Then we reduce this model to population protocols augmented with an *abscence detector*. An absence detector is an oracle that gives hints about *which states are not present in the population*. Each process can interact with this special agent (the absence detector) that monitors other agents in the system, and maintains flags for each state of the protocol. The rest of the model is the same as the PP model. All agents, apart from the absence detector, are modeled as *finite-state machines* that run the *same protocol*. Agents interact in pairs according to some *interaction graph* which specifies the permissible interacting pairs, and update their states in the process. *No agent can predict or control its interactions*. Within this framework, we explore the computational capabilities of this new extension, that we call *Population Protocols with Absence Detector* (*AD*), and study its properties on a purely theoretical ground. As we shall see the AD model is computationally stronger than PPs but this is not what sets it apart. A major new feature of this model is its capability to perform *halting computations*, which allows sequential execution of protocols. Note that although we are currently unaware of how to construct such detectors, in the future, our detector may be implemented via a Bulletin Board regarding the existing states (e.g. each device marks its current state in the board, and all devices can read this board). Such Boards can be implemented easily and have been used in the past [Edi86].

2 Other Previous Work

In the population protocol model [AAD+06], n computational agents are passively mobile, interact in ordered pairs, and the temporal connectivity assumption is a *strong global fairness condition* according to which all configurations

that may always occur, occur infinitely often. These assumptions give rise to some sort of structureless interacting automata model. The usually assumed *anonymity* and *uniformity* (i.e. n is not known) of protocols only allow for commutative computations that eventualy stabilize to a desired configuration. Most computability issues in this area have now been established. Constant-state nodes on a complete interaction network (and several variations) compute the *semilinear predicates* [AAER07]. Semilinearity persists up to $o(\log \log n)$ local space but not more than this [CMN$^+$11]. If constant-state nodes can additionally leave and update fixed-length pairwise marks then the computational power dramatically increases to the commutative subclass of **NSPACE**(n^2) [MCS11a]. For a very recent introductory text see [MCS11b]. Finally, our CTS model is different than the cover-times considered in [BBCK10] in that we allow protocols know the cover times and as their cover-times refer to the time for an agent to meet all other agents.

3 Our Results - Roadmap

In Sections 4 and 5, the newly proposed models are formally defined. Subsection 5.1 in particular, defines halting and output stabilizing computations, as well as the classes of predicates that the AD model can compute in both cases. In Section 6, we illustrate the new model with a simple leader election protocol and give some properties of the AD concerning halting computations. Section 7 first establishes the computational equivalence of the CTS and AD models and then deals with the computational power of the latter. In particular, Section 7.1 shows that all semilinear predicates (whose class is denoted by **SEM**) are stably computable by halting ADs. In Section 7.2, several improved computational lower bounds and an upper bound are presented. In particular, it is first shown that the class **HAD**, of all predicates computable by some AD with a unique leader, includes all multiplication predicates of the form $(bN_1^{d_1}N_2^{d_2}\cdots N_k^{d_k} < c)$, where b, c, d_i, k are constants, $b, c \in \mathbb{Z}$ and $d_i, k \in \mathbb{Z}^+$. We do so by constructing an AD (Protocol 2) that performs *iterative computation*. Then in Subsection 7.2 it is shown that halting ADs can compute any predicate whose support (corresponding language on the input alphabet) is decidable by a Turing Machine (TM) of $O(\log n)$ space. This is shown by *simulating a One Way k-Counter Machine* (k-CM) [FMR68, Min61] with halting ADs. Moreover, it is shown that all predicates in **HAD** are stably computable by a TM of $O(\log^2 n)$ space. Finally, some similarities of the AD model with *Multilset Linear Bounded Automata with Detection* (*MLBAD*s) are pointed out and it is established (however, the proof being left for the full paper) that ADs can simulate such automata. In Section 8, we conclude and present potential future research directions.

4 A Cover-Time Service

We equip pairwise-interacting agents with the following natural capability: *swapping states can know when they have covered the whole population*. Note that we

refer to states and not to nodes. A node may possibly not be ever able to interact with all other nodes, however if nodes constantly swap their states then the resulting random walk must be capable of covering the whole population (e.g. in a complete graph the cover time of a random walk is $n \log n$).

We assume a unique leader in the population which jumps from node to node. What we require is that the leader state *knows* when it has passed from all nodes and we require this to hold iteratively, that is after it knows that it has covered the whole population it can know the same for the next walk, and so on. So we just assume a *cover-time service* which is a black-box for the protocol. We call this extension of PPs with leader and a cover-time service the *Cover-Time Service* (*CTS*) model.

Formally, we are given a population of n agents s.t. initially a node u is in state $(l, D, 0)$ while all other nodes are in state $\perp$. What we require is that $D \in \mathbb{N}$ satisfies the following. If in every interaction (v, w), s.t. the state of v is (l, D, i) and the state of w is $\perp$, v updates to $\perp$ and w to $(l, D, i + 1)$ (swapping their states and increasing i by one) if $D > i + 1$ and to $(l, D, 0)$ otherwise, then in every D consecutive steps $s_1, \ldots, s_D$ (where we can w.l.o.g. assume that a single interaction occurs in each step) it holds that $\{z \in V : z \text{ has obtained } l \text{ at least once in the interval } [s_1, s_D]\} = V$. That is D is an upper bound on the time needed for a swapping state (here l) to visit all nodes (called the *cover-time*). The leader state, no matter which node it lies on, can detect the coverage of V when the step/interaction counter i becomes equal to D. We assume that both D and i are only used for coverage detection and not as additional storage for internal computation (nodes keep operating as finite-state machines). Another way to appreciate this is by imagining that all nodes have access to a global clock that ticks every D rounds.

We explore the computability of the CTS model. In particular, we arrive at an exact characterization of its computational power. We do so by reducing the CTS model to an artificial but convenient variant of population protocols that is equipped with a powerful oracle-node capable of detecting the presence or absence of any state from the population. Our oracle model is of particular theoretical interest as it seems that most PP variants equipped with some capability to detect termination, and not only our particular CTS example, may as well reduce to it.

5 Absence Detectors

A *Population Protocol with Absence Detector* (*AD*) is a 7-tuple $(X, Y, Q, I, \omega, \delta, \gamma)$ where X, Y and Q are finite sets and X is the *input alphabet*, Y is the *output alphabet*, Q is a set of *states*, $I : X \to Q$ is the *input function*, $\omega : Q \to Y$ is the output function, δ is the transition function $\delta : Q \times Q \to Q \times Q$ and γ is the *detection transition function* $\gamma : Q \times \{0, 1\}^{|Q|} \to Q$. If $\delta(a, b) = (c, d)$, where $a, b, c, d \in Q$, we call $(a, b) \to (c, d)$ a *transition* and we define $\delta_1(a, b) = c$ and $\delta_2(a, b) = d$. We also call transition any $(q, a) \to c$, where $q, c \in Q$, $a \in \{0, 1\}^{|Q|}$ so that $\gamma(q, a) = c$.

An AD runs on the nodes of an interaction graph $G = (V, E)$ where G is a directed graph without self-loops and multiple edges, V is a *population* of n *agents* plus a single *absence detector* ($n + 1$ entities in total), and E is the set of permissible, ordered interactions between two agents or an agent and the absence detector. An absence detector is a special node whose state is a vector $a \in \{0,1\}^{|Q|}$, called *absence vector*, always representing the absence or not of each state from the population; that is, $q \in Q$ is absent from the population in the current configuration iff $a[q] = 1$. From now on we will denote the absence detector by a unless stated otherwise. Throughout the section we consider only complete interaction graphs, that is all agents may interact with each other and with the absence detector.

Initially, each agent except the absence detector senses its environment (as a response to a global start signal) and receives an input symbol from X. We call an *input assignment* to the population, any string $x = \sigma_1\sigma_2 \ldots \sigma_n \in X^*$, where by n we denote the population size. Then all agents that received an input symbol apply the input function on their symbols and obtain their initial states. Given an input assignment x the absence detector is initialized by setting $a[q] = 0$ for all $q \in Q$ so that $\exists \sigma_k \in x : I(\sigma_k) = q$ and $a[q] = 1$ for all other $q \in Q$.

A *population configuration*, or more briefly a *configuration* is a mapping $C : V \to Q \cup \{0,1\}^{|Q|}$ specifying a state $q \in Q$ for each agent of the population and a vector $a \in \{0,1\}^{|Q|}$ for the absence detector. We call an *initial configuration*, a configuration that specifies the initial state of each agent of the population and the initial absence vector of the absence detector w.r.t. a given input assignment x (as previously described). Let C, C' be two configurations and $u \in V - \{a\}$, $a \in \{0,1\}^{|Q|}$ be an agent and the absence vector of the detector, respectively. We denote by $C(u)$ the state of agent $u \in V$ under configuration C. We say that C *yields* C' *via encounter* $(u, a) \in E$ and denote by $C \xrightarrow{(u,a)} C'$, if $C'(u) = \gamma(C(u), a)$, $C'(w) = C(w)$, $\forall w \in (V - \{u, a\})$ and $C'(a) = a'$ so that $a'[q] = 0$, $\forall q \in Q$ where $\exists w \in V : C'(w) = q$ and $a'[q] = 1$ otherwise. The previous transition can be similarly defined for the reverse interaction (a, u). In addition, given two distinct agents $u, \upsilon \in V$, where $u, \upsilon \neq a$, we say that C *yields* C' *via encounter* $e = (u, \upsilon) \in E$ and denoted by $C \xrightarrow{e} C'$, if $C'(u) = \delta_1(C(u), C(\upsilon))$, $C'(\upsilon) = \delta_2(C(u), C(\upsilon))$, $C'(w) = C(w)$, for all $w \in (V - \{u, \upsilon, a\})$ and $C'(a) = a'$ updated as previously. We say that C can go to C' in one step, denoted $C \to C'$, if $C \xrightarrow{t} C'$ for some $t \in E$. We write $C \xrightarrow{*} C'$ if there is a sequence of configurations $C = C_0, C_1, \ldots, C_k = C'$, such that $C_i \to C_{i+1}$ for all i, $0 \leq i < k$, in which case we say that C' is *reachable* from C.

We call an *execution* any finite or infinite sequence of configurations C_0, C_1, $C_2, \ldots$, where C_0 is an initial configuration and $C_i \to C_{i+1}$, for all $i \geq 0$. The interacting pairs are chosen by an adversary. A strong global *fairness condition* is imposed on the adversary to ensure the protocol makes progress. An infinite execution is *fair* if for every pair of configurations C and C' such that $C \to C'$, if C occurs infinitely often in the execution then so does C'. An adversary scheduler is fair if it always leads to fair executions. A *computation* is an infinite fair

execution. An interaction between two agents is called *effective* if at least one of the initiator's or the responder's states is modified (that is, if C, C' are the configurations before and after the interaction, respectively, then $C' \neq C$).

Note that since X, Y, and Q are finite, the description of an AD is independent from the population size n. Moreover, agents cannot have unique identifiers (uids) since their are unable to store them in their memory. As a result, the AD model preserves both *uniformity* and *anonymity* properties that the basic Population Protocols have.

5.1 Stable Computation

We call a *predicate over* X^* any function $p : X^* \to \{0, 1\}$. p is called *symmetric* if for every $x \in X^*$ and any x' which is a permutation of x's symbols, it holds that $p(x) = p(x')$ (in words, permuting the input symbols does not affect the predicate's outcome). In this work we are interested in the computation of symmetric predicates.

A configuration C is called *output stable* if for every configuration C' that is reachable from C it holds that $\omega(C'(u)) = \omega(C(u))$ for all $u \in V$, where $\omega(C(u))$ is the output of agent u under configuration C. In simple words, no agent changes its output in any subsequent step and no matter how the computation proceeds. We assume that a is the only agent that does not have an output. So the output of the population concerns only the rest of the agents.

A predicate p over X^* is said to be *stably computable* by the AD model, if there exists a AD $\mathcal{A}$ such that for any input assignment $x \in X^*$, any computation of $\mathcal{A}$ on a complete interaction graph of $|x| + 1$ nodes beginning from the initial configuration corresponding to x reaches an output stable configuration in which all agents except a output $p(x)$.

The existence of an absence detector allows for halting computations. An AD $\mathcal{A} = (X_{\mathcal{A}}, Y_{\mathcal{A}}, Q_{\mathcal{A}}, I_{\mathcal{A}}, \omega_{\mathcal{A}}, \delta_{\mathcal{A}}, \gamma_{\mathcal{A}})$, is *halting* if there are two special subsets $Q_{h\_accept}, Q_{h\_reject} \subseteq Q_{\mathcal{A}}$, in which any agent stops participating in effective interactions (halts), giving output $1, 0$ respectively. We say that a predicate p over X^* is *computable by a halting AD* $\mathcal{A}$ if for any input assignment $x \in X^*$, any computation of $\mathcal{A}$ on a complete interaction graph of $|x| + 1$ nodes beginning from the initial configuration corresponding to x reaches an output stable configuration in which, *after a finite number of interactions*, all agents, except for a, are in states of Q_{h_accept} if $p(x) = 1$ and of Q_{h_reject} otherwise.

Let $\mathbf{SPACE}(f(n))$ ($\mathbf{NSPACE}(f(n))$) be the class of languages decidable by some (non) deterministic TM in $O(f(n))$ space. For any class $\mathbf{L}$ denote by $\mathbf{SL}$ its commutative subclass. In addition, we denote by $\mathbf{SEM}$, the class of the semilinear predicates, consisting of all predicates definable by first-order logical formulas of Presburger arithmetic (see, e.g., [GS66]).

6 Examples and Properties

We begin with a leader-election AD. $X = \{1\}$, $Q = \{l, f, q_{halt}\}$, $I(1) = f$, δ is defined as $(l, f) \to (l, q_{halt})$, and γ as $(f, a) \to l$, if $a[l] = 1$ and $(l, a) \to q_{halt}$, if

$a[f] = 1$. Note that both the output alphabet and the output function are not specified since the output is meaningless in this setting. The interactions that are not specified in δ and γ are ineffective.

Proposition 1. *The above protocol is a leader election AD.*

Proof. It is easy to see that a leader is initially generated that halts the non-leaders computation. The leader halts once it is informed by the absence detector that all non-leaders have halted. □

The following are some interesting properties of the AD model.

Proposition 2. *Any AD with stabilizing states has an equivalent halting AD.*

Proof. $\{0,1\}^{|Q|}$ can be partitioned into a state stable subset and a state unstable subset. $a \in \{0,1\}^{|Q|}$ is state stable iff for all $q_1, q_2 \in Q$ (not necessarily distinct) such that $a[q_1] = a[q_2] = 0$, $\delta(q_1, q_2) = (q_1, q_2)$ and $\gamma(q_1, a) = q_1$. If we let all agents know in advance the above partitioning (note that this is constant information, so storing it is feasible) then we have the required termination criterion; that is, an agent halts iff it encounters a detector with a state stable absence vector. □

From now on, we only consider ADs that halt.

A very interesting feature of ADs is that they can be sequentially composed. This means that given two ADs $\mathcal{A}$ and $\mathcal{B}$ we can construct a AD $\mathcal{C}$ which has the input of $\mathcal{A}$ and the output of $\mathcal{B}$ given $\mathcal{A}$'s output as input. First, $\mathcal{C}$ runs as $\mathcal{A}$ on its inputs and once the absence detector detects $\mathcal{A}$'s halt, $\mathcal{C}$ starts $\mathcal{B}$'s execution on using the output of $\mathcal{A}$ as input. The next theorem exploits the sequential composition of ADs to show that any AD can assume the existence of a unique leader.

Proposition 3. *Any AD $\mathcal{A}$ has an equivalent AD $\mathcal{B}$ that assumes a unique leader which does not obtain any input.*

Proof. For the one direction, $\mathcal{B}$ may trivially simulate $\mathcal{A}$ by ignoring the leader. Then for all computations of $\mathcal{A}$ on n agents there is an equivalent computation of $\mathcal{B}$ on $n+1$ agents. For the other direction, $\mathcal{A}$ first elects a unique leader and then simulates $\mathcal{B}$ by considering the input of the agent that has been elected as a leader as a "virtual" agent. The leader creates a bit which moves between the non-leaders. Whenever the leader encounters the bit it interacts with the virtual agent that it carries in its own state. The role of the leader in the "virtual" interaction, that is, whether it is the initiator or the responder can be determined by its role in the real interaction in which it encountered the bit. Note that $\mathcal{B}$'s computations on $n+1 \geq 3$ agents are simulated by $\mathcal{A}$ on n agents. □

Based on this fact, we only consider ADs that assume the existence of such a unique leader in the initial configuration that is responsible for all effective interactions (non-leader interactions do not cause state modifications). We denote by **HAD** *the class of all predicates computable by some AD with a unique leader.*

6.1 The Power of 2 Protocol

We now construct an AD that computes the non-semilinear predicate $(N_1 = 2^d)$, which is true if the number of 1s in the input is a power of 2 (Protocol 1). This protocol illustrates the ability of ADs to perform iterative computations (which was impossible in PP model).

Protocol 1. *Power of 2*

1: $X = \{1\}, Q = (\{l\} \times \{q_0, q_1, q_2, q_3, q_4\}) \cup (\{n\} \times \{1, \bar{1}, 1'\}) \cup \{q_{accept}, q_{reject}\}$,
2: $I(1) = (n, 1)$ only for the non-leaders,
3: the leader is initialized to (l, q_0),
4: δ:

$$
\begin{aligned}
(l, q_0), (n, 1) &\to (l, q_1), (n, \bar{1}) \\
(l, q_1), (n, 1) &\to (l, q_2), (n, \bar{1}) \\
(l, q_2), (n, 1) &\to (l, q_3), (n, 1') \\
(l, q_3), (n, 1) &\to (l, q_2), (n, \bar{1}) \\
(l, q_4), (n, 1') &\to (l, q_4), (n, 1)
\end{aligned}
$$

5: γ:

$$
\begin{aligned}
(l, q_2), a &\to q_{accept}, \text{ if } a[n, 1] = a[n, 1'] = 1 \\
&\to (l, q_4), \text{ if if } a[n, 1] = 1 \text{ and } a[n, 1'] = 0 \\
(l, q_3), a &\to q_{reject}, \text{ if } a[n, 1] = 1 \text{ and } a[n, 1'] = 0 \\
(l, q_4), a &\to (l, q_1), \text{ if } a[n, 1'] = 1
\end{aligned}
$$

7 Computational Power

We now explore the computational power of the CTS model via the AD model. In particular, we provide several lower bounds and an upper bound for the class **HAD**. By Theorem 1, that we just present, these results carry over to the class of languages computable by CTS protocols.

Theorem 1. *The CTS model is computationally equivalent to the leader-AD model.*

Proof. The CTS-leader may form an absence vector by walking around and keeping track of present states until it covers the whole population. The AD-leader detects the completion of a covering by marking all nodes that it meets and asking the absence detector whether all nodes have been marked. □

7.1 PPs vs. ADs

In [AAER07], they defined the k-truncate of a configuration $c \in \mathbb{N}^Q$ as $\tau_k(c)[q] := \min(k, c[q])$ for all $q \in Q$.

Lemma 1. *For all finite k and any initial configuration $c \in \mathbb{N}^Q$, there is an AD that aggregates in one agent $\tau_k(c)$.*

Proof. The unique leader is aware of the finite bound k and initiates a $|Q|$-vector full of zeros except for a 1 in the position of its own state (note that since the leader election protocol is halting we are allowed to first elect a leader and then execute a second procedure based on the assumption of a leader). When a leader interacts with a non-leader, then the non-leader halts and if the leader's counter corresponding to the non-leader's state was less than k, then the leader increments it by one. The leader halts when the absence detector informs it that non-leaders are absent. □

Theorem 2. SEM $\subseteq$ HAD.

Proof. It was proved in [AAER07] that, for any PP with stabilizing outputs, there exists a finite k such that a configuration is output stable iff its k-truncate is output stable (and the output values are preserved). We let the AD know the k corresponding to the simulated PP. The AD-leader performs a constant number of simulation steps, e.g. k, and then does the following. It marks all non-leaders one after the other, while gathering the k-truncate of their current configuration c. When the detector informs the leader that no unmarked non-leaders have remained, the leader checks whether $\tau_k(c)$ is output-stable (since k is finite and independent of the population size, we may as in Proposition 2 assume that the leader knows in advance the subset of output stable k-truncates). If it is, then c must also be output stable and the protocol halts. If not, then neither is c and the leader drops this truncate, restores one after the other all non-leaders and when no marked non-leader has remained it continues the PP's simulation for another constant number of steps, and so on. □

Taking into account Theorem 2 and the non-semilinear power of 2 predicate (Protocol 1) we have that **SEM** $\subsetneq$ **HAD**.

7.2 Better Lower Bounds and an Upper Bound

We construct now an AD that computes the predicate $(bN_1^{d_1}N_2^{d_2}\cdots N_k^{d_k} < c)$, where b and c are integer constants and d_i and k are nonnegative constants. We again make w.l.o.g. the assumption of a unique leader, and for further simplification we forget about the leader's input.

To simplify the description we first present an AD (Protocol 2) that computes $(bN_1^d < c)$. Define $[c] := \{0, 1, \ldots, |c|\}$ if $c < 0$ and $[c] := \{-c, -c+1, \ldots, 0\}$ if $c \geq 0$. Define u_{-i} to be the subvector of a vector u consisting of all components of u except from component i. We write a vector u as (j, u_{-i}) when we want to

Protocol 2. *VarPower*

1: $X = \{s_1\}, Q = (\{l_1, l_2, \ldots, l_d, l_1^e, l_2^e, \ldots, l_d^e\} \times [c]) \cup \{0,1\}^d \cup \{q_{accept}, q_{reject}\}$,
2: $I(s_1) = 0^d$,
3: the initial state of the leader is $(l_1, -c)$,
4: δ:

$$\begin{aligned}
(l_i, w), (0, u_{-i}) &\to (l_{i+1}, w), (1, u_{-i}), \text{ if } i < d \\
&\to q_{accept}, \text{ if } i = d \text{ and } c \geq 0, w + b \leq -c \text{ or } c < 0, w + b < 0 \\
&\to q_{reject}, \text{ if } i = d \text{ and } c \geq 0, w + b \geq 0 \text{ or } c < 0, w + b \geq -c \\
&\to (l_i, w + b), (1, u_{-i}), \text{ if } i = d \text{ and } c \geq 0, -c \leq w + b < 0 \text{ or} \\
&\qquad\qquad c < 0, 0 \leq w + b < |c| \\
(l_i^e, w), (1, u_{-i}) &\to (l_i^e, w), (0, u_{-i})
\end{aligned}$$

5: γ:

$$\begin{aligned}
(l_i, w), a &\to (l_i^e, w), \text{ if } a[0, u_{-i}] = 1 \text{ and } i > 1 \\
&\to q_{accept}, \text{ if } a[0, u_{-i}] = 1, i = 1 \text{ and } w < 0 \\
&\to q_{reject}, \text{ if } a[0, u_{-i}] = 1, i = 1 \text{ and } w \geq 0 \\
(l_i^e, w), a &\to (l_{i-1}, w), \text{ if } a[1, u_{-i}] = 1
\end{aligned}$$

emphasize that component i of u has the value j. Given an absence vector a, $a[j, u_{-i}] = 1$ is true iff (j, u_{-i}) is absent from the population for all u_{-i}.

We now extend the above construction to devise an AD for the predicate $(bN_1^{d_1} N_2^{d_2} \cdots N_k^{d_k} < c)$, where b and c are integer constants and k is a nonnegative constant. The idea is simple. The leader now holds a k-vector of vectors, l, where l_i is a d_i-vector of states, similar to those of Protocol 2, in order to execute k copies of Protocol 2. The leader still holds a unique counter initialized to $-c$. Similarly, each agent has k components, one for each subprotocol. The AD, in fact, produces all possible assignments of states to l_{ij}. Initially, one step of each subprotocol is executed, then all steps of subprotocol k is executed, then k is reinitialized, $k-1$ is proceeded for one step and again all possible steps of k are executed, when all possible combinations of $k-1$ and k have been exhausted, $k-2$ proceeds for one step, and all possible combinations of $k-1$ and k are reproduced, and so on. After each step, except for the first $k-1$ steps, the terminating conditions of Protocol 2 are checked and if no one is satisfied b is added to the leader's counter.

Finally, by exploiting the above constructions we devise an AD that computes the predicate $\sum_{d_1,d_2,\ldots,d_k=0}^{l} a_{d_1,d_2,\ldots,d_k} N_1^{d_1} N_2^{d_2} \cdots N_k^{d_k} < c$, where $a_{d_1,d_2,\ldots,d_k}$ and c are integer constants and l and k are nonnegative constants. Here, a difference to the previous protocol is that we have many copies of it running in parallel, their number being equal to the number of nonzero coefficients, and each one of them adds to the counter its own coefficient $a_{d_1,d_2,\ldots,d_k}$.

A key difference is that the counter bounds are now set to $-s, s$, where $s := \max(\max_{d_1,d_2,\ldots,d_k=0,\ldots,l} |a_{d_1,d_2,\ldots,d_k}|, |c|)$, and that when we say "in parallel" we can implement this in a round-robin fashion, and let the protocol terminate when no subprotocol can proceed without exceeding the bounds. Then the halting decision simply depends on whether the leader's counter is negative or not. We conclude with the following lower bound on **HAD**.

Theorem 3. *Any predicate of the form* $\sum_{d_1,d_2,\ldots,d_k=0}^{l} a_{d_1,d_2,\ldots,d_k} N_1^{d_1} N_2^{d_2} \cdots N_k^{d_k} < c$, *where* $a_{d_1,d_2,\ldots,d_k}$ *and* c *are integer constants and* l *and* k *are non-negative constants, is in* **HAD**.

Simulating a Counter Machine. In this Section, we prove that ADs and *one-way (online) counter machines* (CMs) [FMR68, Min61] can simulate each other.

The *space* required by a CM in processing its input is the maximum value that any of its counters obtains in the course of the computation. A language $L \subseteq \Sigma^*$ is said to be *CM-decidable in* $O(f(n))$ *space* if some CM which operates in space $O(f(n))$ accepts any $w \in L$ and rejects any $w' \in \Sigma^* \backslash L$. Let **CMSPACE**$(f(n))$ (**NCMSPACE**$(f(n))$ for nondeterministic CMs) be the class of all languages that are CM-decidable in $O(f(n))$ space. Recall that by **SCMSPACE**$(f(n))$ (**SNCMSPACE**$(f(n))$) we denote its symmetric subclass. The following well-known theorem states that any CM of space $O(f(n))$ can be simulated by a TM of space $O(\log f(n))$ and conversely.

Theorem 4 ([FMR68]). **CMSPACE**$(f(n)) =$ **SPACE**$(\log f(n))$ *and* **NCMSPACE**$(f(n)) =$ **NSPACE**$(\log f(n))$.

The above result can also be found as Lemma 3, page 94, in [Iba04].

Corollary 1. **SCMSPACE**$(f(n)) =$ **SSPACE**$(\log f(n))$ *and* **SNCMSPACE**$(f(n)) =$ **SNSPACE**$(\log f(n))$.

We are now ready to establish our final bounds on **HAD**.

Theorem 5. **SSPACE**$(\log n) =$ **SCMSPACE**$(n) \subseteq$ **HAD** $\subseteq$ **SNSPACE**$(\log n) \subseteq$ **SSPACE**$(\log^2 n)$.

Proof. For the lower bound, we show that ADs can simulate CMs (**SSPACE**$(\log n) =$ **SCMSPACE**(n) is from [FMR68]). The CM consists of a control unit, an input terminal, and a constant number of counters. The AD simulates the control unit by its unique leader, which is responsible for carrying out the simulation. The input terminal is formed by the actual input slots of the agents. The k counters are stored by creating a k-vector of bits in the memory of each agent. In this manner, each counter is distributed across the agents. The value of the ith counter at any time is determined by the number of 1s appearing in the ith components of the agents. Since the number of agents is equal to the number of input symbols the space of each counter is linear to the input size

(in fact, we can easily make this $O(n)$ by allowing c bits in each component instead of just one). To take a step, the CM reads or not the next symbol from the input and the sign (0 or positive) of each tape and then, if it read the input, moves to the next input symbol and updates the contents of the counters. The leader of the AD waits or not to encounter an agent whose input is not erased (unread), in the former case erases that input symbol, and waits to encounter the absence detector to learn the set of zero counters. When the latter happens, the leader obtains a vector of -1s, 0, and 1, representing the value to be added to each counter. From that point on, the leader adds these values wherever possible until all of them have been added. Then the leader continues the simulation as above. The proof for the upper bound is similar to the one of Theorem 15 in [AAD$^+$06]. We construct a TM that, starting from any initial configuration, nondeterministically guesses all reachable configurations and always stores at most one. We also invoke Savitch's theorem [Sav70]. □

In the full paper, we also prove that ADs can simulate *Multiset Linear Bounded Automata with Detection* (*MLBAD*) [CVMVM01, Vas08]. This implies that ADs can compute any language produced by random context grammars. Finally, we establish that nondeterministic ADs are computationally equivalent to the deterministic ones.

8 Conclusions

In this work, we proposed the CTS model a new extension of the PP model of Angluin *et al.* that additionally assumes the existence of a cover-time service. By reduction to the absence detector oracle model we were able to investigate and almost completely characterize the computational power of the new model. The introduced global knowledge enables CTSs to perform halting computations, a feature that was missing from the PP model. We explored the properties and the computability of the new model and focused more on halting computations. We showed that all predicates in $\mathbf{SSPACE}(\log n)$ are also in **HAD** and that the latter is a subset of $\mathbf{SSPACE}(\log^2 n)$.

Many interesting questions remain open. The bounds given in this work for halting ADs are not tight. An exact characterization of **HAD** is still elusive. In addition, what happens in the case where the detector does not correctly detect the existing states in the population? Do the protocols presented here work correctly in the case of an adversarial detector? In addition, how is the computability of graph properties of the interaction graph affected by the absence detectors presence? Finally, can one simplify the proof of the upper bound of PPs [AAER07] by simulating them by a one-way 1-CM or by a nondeterministic pushdown automaton?

Acknowledgements. We would like to particularly thank James Aspnes for bringing to our attention some similarities of our model to the eventual leader detector Ω of [FJ06] and also thank Stavros Nikolaou for his useful comments throughout the preparation of this work.

References

[AAC+05] Angluin, D., Aspnes, J., Chan, M., Fischer, M.J., Jiang, H., Peralta, R.: Stably Computable Properties of Network Graphs. In: Prasanna, V.K., Iyengar, S.S., Spirakis, P.G., Welsh, M. (eds.) DCOSS 2005. LNCS, vol. 3560, pp. 63–74. Springer, Heidelberg (2005)

[AAD+06] Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distributed Computing, 235–253 (2006)

[AAE08] Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distributed Computing 21(3), 183–199 (2008)

[AAER07] Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distributed Computing 20(4), 279–304 (2007)

[BBCK10] Beauquier, J., Burman, J., Clement, J., Kutten, S.: On utilizing speed in networks of mobile agents. In: Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, PODC 2010, pp. 305–314. ACM, New York (2010)

[CMN+11] Chatzigiannakis, I., Michail, O., Nikolaou, S., Pavlogiannis, A., Spirakis, P.G.: Passively mobile communicating machines that use restricted space. Theor. Comput. Sci. 412(46), 6469–6483 (2011)

[CVMVM01] Csuhaj-Varjú, E., Martín-Vide, C., Mitrana, V.: Multiset Automata. In: Calude, C.S., Pun, G., Rozenberg, G., Salomaa, A. (eds.) Multiset Processing. LNCS, vol. 2235, pp. 69–84. Springer, Heidelberg (2001)

[Edi86] Edighoffer, J.L.: Distributed, replicated computer bulletin board service. PhD thesis, Stanford, CA, USA, UMI order no. GAX86-19742 (1986)

[FJ06] Fischer, M., Jiang, H.: Self-stabilizing Leader Election in Networks of Finite-State Anonymous Agents. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 395–409. Springer, Heidelberg (2006), http://dx.doi.org/10.1007/11945529_28

[FMR68] Fischer, P.C., Meyer, A.R., Rosenberg, A.L.: Counter machines and counter languages. Mathematical Systems Theory 2(3), 265–283 (1968)

[GS66] Ginsburg, S., Spanier, E.H.: Semigroups, presburger formulas, and languages. Pacific Journal of Mathematics 16, 285–296 (1966)

[Iba04] Ibarra, O.H.: On the computational complexity of membrane systems. Theor. Comput. Sci. 320, 89–109 (2004)

[MCS11a] Michail, O., Chatzigiannakis, I., Spirakis, P.G.: Mediated population protocols. Theor. Comput. Sci. 412, 2434–2450 (2011)

[MCS11b] Michail, O., Chatzigiannakis, I., Spirakis, P.G.: New Models for Population Protocols. In: Lynch, N.A. (ed.) Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool (2011)

[Min61] Minsky, M.L.: Recursive unsolvability of post's problem of "tag" and other topics in theory of turing machines. The Annals of Mathematics 74(3), 437–455 (1961)

[Sav70] Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. J. Comput. System Sci. 4(2), 177–192 (1970)

[Vas08] Vaszil, G.: Multiset grammars, multiset automata, and membrane systems. In: Colloquium on the Occasion of the 50th Birthday of Victor Mitrana, pp. 1–10. Springer (2008)