Clocked Population Protocols

James Aspnes*
Yale University, Department of Computer Science.
james.aspnes@gmail.com

ABSTRACT

Population protocols are required to converge to the correct answer, and are subject to a fairness condition that guarantees eventual progress, but generally have no internal mechanism for detecting when this progress has occurred. We define an extension to the standard population protocol that provides each agent with a clock signal that indicates when the agent has waited long enough. To simplify the model, we represent "long enough" as an infinite time interval, and treat a clocked population protocol as operating over transfinite time. This gives a clean theoretical model that we show how to translate back into finite real-world executions where the clock ticks whenever the underlying protocol is looping or stuck.

Over finite time intervals, the protocol behaves as in the standard model. At nonzero limit ordinals ω, $\omega \cdot 2$, etc., corresponding to clock ticks, the protocol switches to a limit of previous configurations supplemented by an signal registering in an extra component in some of the agents' states. Using transfinite times means that we can represent fairness over sequences of transitions that may include clock ticks with the same definition as over smaller intervals. Using arbitrary ordinals allows using times like ω^2 or ω^3 to represent convergence that depends on detecting convergence repeatedly at lower levels.

We show that a clocked population protocol running in less than ω^k time for any fixed $k \geq 2$ is equivalent in power to a nondeterministic Turing machine with space complexity logarithmic in the size of the population. A consequence of this equivalence is that any symmetric predicate that can be computed in less than ω^k time can be computed in less than ω^2 time, which requires only finitely many clock ticks.

1 INTRODUCTION

A **population protocol** [1] consists of a collection of finite-state agents that interact in pairs. If the scheduling of these interactions is random, population protocols or the closely-related model of chemical reaction networks can, with high probability, perform computations limited only by the number of distinct configurations of the population as a whole [2, 16]. But in the standard model,

scheduling is adversarial, and as a result standard population protocols with a complete interaction graph can only compute predicates definable in first-order Presburger arithmetic [3], a restriction that allows addition, parity, and majority, but that removes even such basic operations as multiplication.

The fundamental limitation of population protocols that yields this result is that while a population protocol is only required to converge to the correct answer in the limit, individual agents cannot determine when this convergence occurs. This makes it difficult to compose population protocols sequentially, and puts even such basic programming tools as nested loops out of reach. One of the key tools in showing that randomized population protocols can perform more sophisticated computations is a **phase clock** that allows the population to detect, with high probability, when enough time has elapsed that its current task is complete.

An explicit mechanism used to detect termination is the **absence detector** of Michail and Spirakis [14]. The absence detector acts like an extra agent in the population, that provides a bit-vector indicating the presence or absence of each other state in the population. Michail and Spirakis show that an absence detector can be implemented by a weaker object called a **cover-time service**, which provides an upper bound on the cover time of a random walk, allowing an agent to deduce when it has successfully encountered every other agent in the system. They show that with a cover-time service, the power of a population protocol with n agents lies between $\mathbf{SPACE}(\log n)$ and $\mathbf{NSPACE}(\log n)$, when both are restricted to symmetric inputs. We can think of this cover-time service, like the internal clocks of randomized population protocols and chemical reaction networks, as instances of a family of possible clock mechanisms whose function is to tell a protocol when it has waited long enough to converge.

We propose an extension to the population protocol model that makes such clocks explicit without committing to any specific implementation: a **clock** oracle that signals to one or more agents when the population has either converged or started looping, a condition formally modeled by reaching a configuration that will recur infinitely often in the future. Such **clocked population protocols** are capable of carrying out any computation that is feasible for a Turing machine with equivalent storage capacity: specifically, with a population of size n, whose state can be represented in $\Theta(\log n)$ space, a clocked population protocol can compute any symmetric predicate on the agents' initial states that can be computed by a nondeterministic Turing machine that also uses $\Theta(\log n)$ space

An advantage of showing that clocked population protocols have power equivalent to **NL** is that we can use closure results on **NL** to program clocked population protocols. For example, the Immerman-Szelepcsényi Theorem [12, 17] and the resulting collapse of the logspace hierarchy mean that clocked population protocols can not only compute **NL** predicates, but can effectively use **NL** computations as subroutines. We use this fact to demonstrate that many

natural extensions of the clocked population protocol model turn out to be equivalent in power.

An issue that arises in clocked population protocols is how to keep track of time. At the lowest level, we still have **protocol transitions** in which individual pairs of agents encounter one another and update their state as defined by the standard transition relation. But at a higher level, we have **clock transitions** where some of the agents in a configuration receive **ticks** that indicate that a clock cycle has completed. We would like to have a method for indexing events in a protocol that clearly distinguishes between protocol transitions and clock ticks. The method we use is to assign each event to an **ordinal number**, a generalization of the natural numbers that includes transfinite elements.

In this representation, the ordinary passage of time is represented by finite intervals, while clock ticks occur at limit ordinals ω, $\omega \cdot 2$, $\omega \cdot 3$, etc. A typical time for an event might be something like $\omega \cdot 3 + 28$. This is an event that follows 3 clock ticks—each arriving after what might a very long sequence of ordinary protocol transitions—followed by 28 ordinary protocol transitions. By using ordinals, we allow for the possibility of introducing "higher-order" clock ticks, indicating longer intervals ω^2, ω^3, and so forth, during which even infinitely many lower-order clock ticks still leave the protocol stuck. The use of ordinal arithmetic for this purpose allows us to define a consistent rule for what configurations are eligible to appear at these times (essentially any configuration that occurs without bound in the interval leading up to the limit ordinal), as well as a straightforward extension of the usual global fairness condition for standard population protocols to clocked population protocols.

Though the transfinite-execution model is not something one could reasonably expect to implement in practice, we show that it is equivalent to a finite-execution model where clock ticks are delivered once the protocol reaches a terminal strongly-connected component in an appropriately-defined graph. In addition to giving a practical rule for deciding when to send in a clock tick, this also shows that output-stable clocked population protocols compute symmetric functions in **P**, and that output-stable clocked population protocols that finish in less than ω^k time for any fixed, finite k, compute symmetric functions in **NL**. Together with the converse result on simulating **NL**, this in fact shows that a transfinite clock hierarchy based on counting layers of increasingly patient clocks never gets off the ground: any protocol that finishes in less than ω^k time computes the same function as a protocol that finishes after receiving finitely many first-order clock ticks.

The goal of this work is to explore the effect of adding clocks to population protocols, and to this end we consider primarily the simplest model with a complete interaction graph and a generic clock. We consider some variants on the clock mechanism and show that for the most part they compute the same functions. In the conclusion, we briefly discuss additional possible extensions of the model.

1.1 Other related work

Adding a clock is not the only way to increase the power of the standard population protocol model. The **community protocol**

model of Guerraoui and Ruppert [10], which allows agents to remember the identities of a constant number of other agents, boosts the power of a system with n agents to decide any language in **NSPACE**$(n \log n)$. The still stronger **mediated population protocol** model of Chatzigiannakis *et al.* [8], which stores information on edges between agents, boosts this power still further to **NSPACE**(n^2) [6]. It is also possible to expand the space in each agent, which is $O(1)$ in the standard model. Chatzigiannakis *et al.* [7] have shown that even a modest extension to $\Theta(\log n)$ bits per agent allows computations of symmetric predicates in **NSPACE**$(n \log n)$ (as in the community protocol model), while smaller extensions up to $o(\log \log n)$ still limit population protocols to the semilinear predicates computable in the standard model.

Unfortunately, the substantial computational power of these extensions argues against their practical application. In contrast, the merely **NL** ⊆ **P** power of previous mechanisms for detecting termination inside a population protocol or similar model [2, 14, 16] suggests that such mechanisms are more likely to be realizable in a physical system that runs for a reasonable amount of time. We consider the present work to be firmly in this tradition, even though for purposes of analysis we find it convenient to model the intervals between clock ticks using transfinite time.

The idea of modeling computation over transfinite time has precedent in work on transfinite Turing machines [11] as a model for **supertasks** [18], a concept extensively studied in the philosophical literature. A supertask is a task that involves an infinite number of steps (often taken as occurring over decreasing time intervals whose sum converges to a finite bound), and the problem of characterizing sensible outcomes of supertasks goes as far back as Zeno's Paradox. In a sense, clocked population protocols are carrying out supertasks, but our choice for the behavior of these systems in the limit is less open to controversy, because it is implied by the goals of the model. Our purpose is only to represent waiting long enough that further waiting will have no effect on the possible outcomes of a protocol. Were we actually modeling infinite computations, a different limit definition (for example, taking the limit of each agent's state separately) might be more appropriate.

1.2 Ordinals and ordinal arithmetic

In this section, we give a brief overview of the **ordinal numbers**, a generalization of the natural numbers that include infinite values. More details on the ordinals can be found in any textbook on set theory, for example in Chapter 2 of Jech [13].

Formally, an ordinal number is an equivalence class over totally-ordered sets $(S, \leq)$ that are **well-ordered**, meaning that every subset T of S has a least element. A standard construction due to von Neumann [19] represents each ordinal as the set of all smaller ordinals, so that 0 is represented by the empty set $\emptyset$, $1 = \{0\}$, $2 = \{0, 1\}$. The finite ordinals $0, 1, 2, \ldots$ are the **natural numbers**. The first infinite ordinal, denoted ω, is just the set of all finite ordinals $\{0, 1, 2, \ldots\}$.

In this representation, $\alpha \leq \beta$ if α is a subset of β, $\alpha < \beta$ if α is an element of β, the minimum of a set of ordinals is just their common intersection, the **successor** $\alpha + 1$ of an ordinal α is represented by $\alpha \cup \{\alpha\}$, and the supremum of a set of ordinals is their union. For example, $\omega + 1$ is represented by $\{0, 1, 2, \ldots; \omega\}$, $\omega + 2$ by

$\{0, 1, 2, \dots ; \omega, \omega + 1\}$, and so on. Not every ordinal is a successor. An ordinal (like 0 or ω) that is not a successor of any other ordinal is called a **limit ordinal**.

We adopt the usual convention that arbitrary ordinals are denoted by lowercase Greek letters, while finite ordinals are denoted by lowercase Latin letters.

Ordinal arithmetic is defined using operations on the corresponding ordered sets. Addition corresponds to concatenation: the ordinal $\omega + \omega$ consists of two copies of ω laid end-to-end, and is represented in set form by $\{0, 1, 2, \dots ; \omega, \omega + 1, \omega + 2, \dots\}$. Addition involving infinite ordinals is not commutative in general: $1 + \omega = \omega$, since there is a one-to-one map that preserves the order type, but $\omega + 1 \neq \omega$. Note that if $\beta \neq 0$, $\alpha + \beta$ is a limit ordinal if and only if β is.

Multiplication is defined recursively by the rule that $\alpha \cdot 0 = 0$, $\alpha \cdot (\beta + 1) = \alpha \cdot \beta + \alpha$, and, when γ is a limit ordinal, $\alpha \cdot \gamma = \sup_{\beta < \gamma} \alpha \cdot \beta$, which can be represented in set form by $\bigcup_{\beta \in \gamma} \alpha \cdot \beta$.[1] It is also possible to define multiplication using order types by applying lexicographic order to $\alpha \times \beta$, with the least-significant value provided first. Like addition, multiplication is generally not commutative when one or both operands are infinite: for example, $\omega \cdot 2$ consists of two copies of ω laid end-to-end, and is equal to $\omega + \omega$, but $2 \cdot \omega$ consists of ω copies of 2 laid end-to-end, and is equal to ω.

Exponentiation is defined similarly to multiplication, with $\alpha^0 = 1$, $\alpha^{\beta+1} = \alpha^\beta \cdot \alpha$, and $\alpha^\gamma = \sup_{\beta < \gamma} \alpha^\beta$. Note that exponentiation involving infinite ordinals can produce results that look strange compared to what happens with natural numbers: for example, $2^\omega = \omega < \omega^2$ and ω^ω is order-isomorphic to the set of all *finite* sequences of natural numbers ordered first by increasing size and then lexicographically (or to the set of all infinite sequences that are eventually all 0). Both 2^ω and ω^ω are countable.

As with finite ordinals, division by nonzero ordinals is possible. Specifically, if α and β are ordinals, and $\beta \neq 0$, then there are unique ordinals γ and ρ such that $\alpha = \beta \cdot \gamma + \rho$ and $\rho < \beta$.

Cantor's Normal Form Theorem (see [13, Theorem 2.26]) says that any ordinal α has a unique representation $\alpha = \omega^{\beta_1} \cdot k_1 + \cdots + \dots \omega^{\beta_n} \cdot k_n$, where n is finite, $\alpha \geq \beta_1 > \beta_2 > \cdots > \beta_n$ are ordinal numbers, and $k_1, \dots, k_n$ are nonzero natural numbers. In effect, the normal form theorem says that any ordinal number can be represented as a sequence of finite coefficients indexed by larger and larger powers of ω, which for our purposes will represent increasing durations of waiting for convergence. Most of the ordinal numbers we will be dealing with in this work will be small, with representations typically of the form $\omega \cdot k + \ell$. But occasionally it will be useful to consider larger ordinals.

2 MODEL

We use a variant of the standard population protocol model that extends the state of each agent to include an extra flag to signal a clock tick.

In the standard model [1], a population protocol is described by a tuple $\langle X, Y, Q, I, O, \delta \rangle$, where X and Y are the input and output alphabets, Q is the state space for agents, $I : X \to Q$ and $O : Q \to Y$

---

[1] This is an example of **transfinite recursion**, which generalizes ordinary recursion by requiring a rule for handling limit ordinals. Typically a recursively-defined value at a limit ordinals will itself be a limit (supremum in this case) of values at smaller ordinals.

---

are functions translating inputs to states and extracting outputs from states, and $\delta : Q \times Q \to Q \times Q$ is a **transition function** used to update the states of two agents that interact with each other. A **population** consists of n agents organized as the nodes of a directed **interaction graph**. A **configuration** specifies the state of each agent. A **step** consists of taking two agents u and v such that uv is an edge in the interaction graph, and updating their states according to δ. It is assumed that which pair of agents interact at each step is controlled by an **adversary**, but the adversary is restricted the **global fairness condition** that if some configuration C_1 occurs infinitely often, and there is a step that transforms C_1 into C_2, then C_2 also occurs infinitely often.

In the present work, we will assume that the interaction graph is complete: any agent may interact with any other agent at any time. But our extension to the standard model applies equally well to a more restricted interaction graph.

2.1 Adding the clock

We would like to add a mechanism for detecting the passage of time to this model, in the form of an oracle that provides extra information to the agents. We do so by extending the state of each agent to include a clock bit that is provided as input to δ and that may be set to 1 by the external clock. By convention, we will not allow a protocol to use the clock bits for its own purposes. We enforce this by requiring that δ always outputs states in which the clock bits are set to 0.

This effectively gives δ the type $Q \times \{0, 1\} \times Q \times \{0, 1\} \to Q \times Q$, where the extra bits represent the clock bit at each agent. This is similar to the approach taken by Fischer and Jiang in their work on self-stabilizing leader election with the $\Omega?$ oracle [9]. However, changes to the clock bits are more restricted than in this and subsequent work applying oracles to population protocols (e.g., [4, 5]). The role of the clock bit on an agent is to act as a inbox for clock ticks, set by special clock transitions that occur only when the protocol is looping or stuck, and reset only by subsequent protocol transitions involving the agent.

To save space, we represent a state in which a clock bit is set typographically using a prime (or "tick") symbol. So a state A does not have the clock bit set, but the corresponding state A' does.

2.2 Clocked executions

The idea of the clock is that the system only delivers a clock tick when the protocol has run long enough that anything that could still happen has already happened.

Applying a simple induction to the global fairness condition shows that if C_1 is a configuration that occurs infinitely often, then any configuration C_2 that is **reachable** from C_1 by zero or more transitions also occurs infinitely often. Because the number of configurations is finite, this means that any execution of a standard population protocol eventually converges to some terminal strongly-connected component (SCC): a set of configurations S such that all configurations in the set are reachable from all of the others, and no transition leaves the set. Once a protocol has reached such a set, no further progress is possible. The standard model says that a protocol **stably computes** a predicate if, in this terminal SCC, all configurations produce the correct output at all agents.

We would like to define the clock to fire only once we have reached a set that is a terminal SCC with respect to protocol transitions. While it would be possible to do so directly, doing so by taking a limit over an infinite execution fragment gives the same result in a way that interacts more cleanly with the fairness condition. Later we will recover a graph-theoretic interpretation of this definition, to make the behavior of clocked population protocols easier to reason about.

Let α be a limit ordinal. A set of times T is **cofinal** in α if, for any $\beta < \alpha$, there is some γ in T such that $\beta \leq \gamma < \alpha$. For example, the times $2, 4, 8, 16, \ldots$ are cofinal in ω. We will use this concept to define the set of configurations that may occur (possibly with clock bits set) at time α.

We say that configurations C_1 and C_2 are **equivalent**, written $C_1 \sim C_2$, if the only difference between them is in the clock bits. We refer to the part of a configuration that omits the clock bits as the **protocol configuration**; two configurations are equivalent if and only if they have the same protocol configurations.

An **execution** Ξ of length α consists of a sequence of configurations C_β indexed by the ordinals $\beta < \alpha$ satisfying certain consistency constraints. A configuration C is **enabled** following Ξ if:

(1) $\alpha = 0$ and C is the initial configuration,
(2) $\alpha = \gamma + 1$ and C follows from an application of the transition function to two agents in the configuration C_γ that appears at time γ, or
(3) α is a nonzero limit ordinal, and the set of times at which configurations equivalent to C occur is cofinal in α.

An execution is **valid** if each configuration C_β is enabled following the prefix of Ξ of length β. Henceforth we will only consider valid executions.

Note that we only consider executions that have a length. This avoids complications that would arise from executions indexed over all of the ordinals (for example, such executions would not be definable as sets in the usual Zermelo-Fraenkel set theory).

The definition of enabled configurations at successor ordinals matches the standard model. The definition at (small) nonzero limit ordinals captures the intuition of waiting long enough to reach a terminal SCC, because the only configurations enabled at time $\omega \cdot k$ are those that appear infinitely often during the preceding standard execution, which are precisely those in the set to which the protocol converges.

Which of the enabled configurations occurs at each nonzero limit ordinal α is chosen by the adversary, the same as at successor ordinals. At limit ordinals, in addition to choosing from all configurations cofinal in α, the equivalence condition allows the adversary to apply clock bits to any subset of the agents, including the empty subset. To make the clock bits useful requires imposing restrictions on this choice, which we do by extending the standard global fairness condition to transfinite intervals.

2.3 Fairness in clocked executions

We define an execution to be **fair** if, for any ordinal α, if the set of times at which a configuration C is enabled is cofinal in α, then C occurs cofinally in α. We define an execution fragment to be fair if

the same condition holds for any α less than or equal to the length of the execution fragment.

A standard execution consists of times less than ω, and restricting attention to this interval yields the standard fairness condition, since on ω, occurring infinitely often and occurring cofinally coincide. But over longer intervals, having C occur infinitely often when it is enabled infinitely often is not enough.

For example, suppose that $C_1 \rightarrow C_2$, and C_1 occurs infinitely often over both the intervals $[0, \omega)$ and $[\omega, \omega \cdot 2)$. We do not wish the clock tick at time ω to change the effect of the fairness condition, but if we only require C_2 to occur infinitely often in the entire execution, it is possible for it to do so only in $[0, \omega)$ (or only in $[\omega, \omega \cdot 2)$), yielding an execution that is unfair by the standard definition within this interval. Cofinality prevents this, and indeed we can show that the extended definition is equivalent to the standard definition over all intervals of length ω:

LEMMA 2.1. *Let Ξ be a fair execution. Then if $C_1 \rightarrow C_2$, and C_1 occurs infinitely often in some interval $[\alpha, \alpha + \omega)$, C_2 also occurs infinitely often in $[\alpha, \alpha + \omega)$.*

PROOF. Let $\alpha + k_1 < \alpha + k_2 < \ldots$ enumerate the times at which C_1 appears in $[\alpha, \alpha + \omega)$. Because the sequence $k_1, k_2, \ldots$ is infinite, it is unbounded in ω, thus $\alpha + k_1, \alpha + k_2, \ldots$ is cofinal with $\alpha + \omega$. So C_2 is enabled at times $a + k_i + 1$, which are also cofinal in $\alpha + \omega$. Fairness then implies that C_2 occurs at times cofinal in $\alpha + \omega$.

Suppose that only finitely many of those times occur in $[\alpha, \alpha + \omega)$. Then there is some largest time $\alpha + k < \alpha + \omega$ at which C_2 occurs. But then there is no time β with $\alpha + k < \beta < \alpha + \omega$ at which C_2 occurs, and C_2 does not occur cofinally in $\alpha + \omega$, a contradiction. So C_2 occurs infinitely often in $[\alpha, \alpha + \omega)$. □

In addition to generalizing the standard definition, our definition also forces the adversary to eventually set the clock bits. Consider an adversary that never sets a clock bit. In the interval $[0, \omega^2)$, at each time $\omega, \omega \cdot 2, \omega \cdot 3, \ldots$, there is some enabled configuration $C'_{\omega \cdot k}$ that includes clock bits. Since there are only finitely many possible such configurations, there is some specific configuration C' that is enabled at infinitely many times of the form $\omega \cdot k$. These times are cofinal in ω^2, so C' must also occur cofinally in ω^2. It follows that in ω^2, the clock ticks infinitely often.

Note that this does not provide any particular guarantee on when the clock bits arrive: just as the adversary may delay a protocol transition for any finite amount of steps, the adversary may similarly delay a particular limit configuration over any finite number of limit times. But it cannot do so forever.

3 TRANSITION GRAPHS

Working directly with transfinite executions is awkward. In this section, we give an alternative characterization of a clocked population protocol's executions based on paths through a directed **transition graph**, where each node in the graph represents a configuration and each edge represents either a transition between configurations or the delivery of a clock tick taking some configuration to an equivalent configuration.

The transition graph is constructed recursively, where each layer G_k of the recursion adds transitions that can occur at times that are a multiple of ω^k. The bottom layer G_0 contains an edge for

all transitions that can occur in one time unit, as the result of applying the protocol's transition relation δ to two agents. We write $C_1 \rightarrow_0 C_2$ if such an edge appears in G_0, and write $C_1 \rightarrow_0^* C_2$ if there is a (nonempty) directed path from C_1 to C_2 in G_0. Similarly, we will write $C_1 \rightarrow_k C_2$ or $C_1 \rightarrow_k^* C_2$ if there is a single edge or directed path, respectively, from C_1 to C_2 in G_k.

To construct G_{k+1} from G_k, we add edges representing the arrival of clock ticks at multiples of ω^k. To do so, we need to detect when some configuration in G_k might occur cofinally in an interval $[\alpha, \alpha + \omega^k)$ in a fair execution.

At minimum, we need $C \rightarrow_k^* C$, so that C can occur infinitely often. But this is not enough: we also need it to be the case that the fairness condition does not drive us to a state C_2 from which we cannot return to C. This essentially means that no such state C_2 can be reachable from C at all, or in other words that C is an element of a terminal strongly-connected component in G_k.

Formally, define $G_{k+1} = (V_{k+1}, E_{k+1})$, where $V_{k+1} = V_k$ is the set of all configurations of the protocol (including clocked configurations), and $E_{k+1} = E_k \cup \{(C, C') \mid C \sim C'$ and C is in a terminal SCC in $G_k\}$. We will refer to the edges in $E_{k+1} \setminus E_k$ as level $k + 1$ edges.

The full transition graph G is just the union of all G_k for $k \in \mathbb{N}$. Note that if we restrict ourselves to configurations of a fixed size, all but a finite number of these graphs are identical, because at some point we run out of edges to add. But for simplicity we will imagine each G_k is an infinite graph that represents all possible population sizes, so G will in general be a limit $\bigcup_{k=0}^{\infty} G_k$. Fortunately, we do not have to go past ω to compute G, since we only consider protocols with finite populations.

THEOREM 3.1. *Given a clocked population protocol, let $G_0, G_1, \ldots$ be the family of transition graphs defined as above, and consider some fair execution Ξ.*

The following hold for any configurations C and D:

(1) *If C occurs at time α, and D occurs at time $\beta \in [\alpha, \alpha + \omega)$, then there is a path from C to D in G_k.*
(2) *If C occurs cofinally in Ξ over some interval $[\alpha, \alpha + \omega^k + 1)$, then D occurs cofinally over the same interval if and only if there is a path from C to D in G_k.*
(3) *If D occurs in $[\alpha, \alpha + \omega^{k+1})$, then D occurs cofinally in $[\alpha, \alpha + \omega^{k+1})$ if and only if D is in a terminal strongly-connected component of G_k.*

PROOF. By induction on k.

When $k = 0$, G_0 consists only of transitions that do not involve clock bits, and the interval $[\alpha, \alpha + \omega^{k+1})$ contains only successor ordinals. We can show:

(1) There is a path in G_0 from C to each configuration C_β with $\alpha \le \beta < \alpha + \omega^{k+1} = \alpha + \omega$. The proof is by induction on β. The base case is $\beta = \alpha$; use the empty path. For the induction step, go from β to $\beta + 1$ by extending the path from $C = C_\alpha$ to C_β by the edge from C_β to $C_{\beta+1}$.
(2) If there is no path from C to D in G_0, then let β be some time at which C occurs in $[\alpha, \alpha + \omega)$. Then from the preceding claim, D does not occur in $[\beta, \beta + \omega) = [\beta, \alpha + \omega)$ and in particular there is no time γ such that $\beta < \gamma < \alpha + \omega$ at which D occurs. So D does not occur cofinally in $[\alpha, \alpha + \omega)$.

Alternatively, suppose there is a path from C to D in G_0. We will show that D occurs cofinally in $[\alpha, \alpha + \omega)$ by induction on $d(C, D)$. If $d(C, D) = 0$, then $D = C$ and the result holds trivially. If $d(C, D) > 0$, let B be the last configuration on a shortest path from C to D in G_0. Then by the induction hypothesis, B occurs cofinally in $[\alpha, \alpha + \omega)$. For each time γ at which B occurs, D is enabled at time $\gamma + 1$, so D is enabled cofinally in $[\alpha, \alpha + \omega)$, and because the execution is fair, it must occur cofinally as well.

(3) Let S be the strongly-connected component of G_0 that contains D. If S is not terminal, there is an outgoing edge from S to some configuration B, such that there is no path from B to D (since B is not in the same strongly-connected component). By the preceding claim, if D occurs cofinally in $[\alpha, \alpha + \omega)$, then so does B. But then, since there is no path from B to D, D cannot occur cofinally in $[\alpha, \alpha + \omega)$.

Conversely, suppose there is no outgoing edge from S. From (1), once D occurs at some time $\beta \in [\alpha, \alpha + \omega)$, only configurations B reachable from D can occur in $[\beta, \alpha + \omega) = [\beta, \beta + \omega)$. Since there are only finitely many such B, one of them occurs infinitely often and thus cofinally in $[\beta, \beta + \omega)$. But since B is in the same SCC as D, there is a path from B to D, and so D occurs cofinally in $[\beta, \beta + \omega)$ (and thus $[\alpha, \alpha + \omega)$ by (2)).

For larger k, suppose that the claim holds for $k - 1$. We will essentialy repeat the argument for the base case G_0, but now we must deal with limit ordinals and edges corresponding to limit transitions. Fortunately we can refer to the preceding arguments to handle the successor ordinal cases.

(1) For the first claim, we will again show by induction on $\beta \in [\alpha, \alpha + \omega^{k+1})$ that there is a path in G_k from $C = C_\alpha$ to C_β. The base case $\beta = \alpha$ and the induction step for successor ordinals $\beta + 1$ are the same as for $k = 0$. This leaves the case where β is a limit ordinal.

Expand β as $\alpha + \omega^k \cdot q + \rho$, where the quotient q is finite and the remainder $\rho < \omega^k$. We will first argue by induction on q that there is a path in G_k from C_α to $C_{\alpha + \omega^k \cdot q}$. If $q = 0$, this is trivial. Otherwise, by the induction hypothesis on q there exists a path in G_k from C_α to $C_{\alpha + \omega^k \cdot (q-1)}$. For $C_{\alpha + \omega^k \cdot q}$ to be enabled at $\alpha + \omega^k \cdot q$, there must be a configuration $C^* \sim C_{\alpha + \omega^k \cdot q}$ that occurs cofinally in $[\alpha + \omega^k \cdot (q-1), \alpha + \omega^k \cdot q)$. From the induction hypothesis on G_{k-1}, this can only be the case if C^* appears in a terminal SCC of G_{k-1}, in which case there is an edge in G_k from C^* to $C_{\alpha + \omega^k \cdot q}$. In addition, the occurrence of C^* in this interval also implies that there is a path in G_{k-1} form C_α to C^*. Stitching this path and edge onto the previous path gets us the claimed path in G_k to $C_{\alpha + \omega^k \cdot q}$.

We still have to deal with finding an extension from $C_{\alpha + \omega^k \cdot q}$ to $C_\beta = C_{\alpha + \omega^k \cdot q + \rho}$, but as $\rho < \omega^k$, $\beta \in [\alpha + \omega^k \cdot q, \alpha + \omega^k \cdot q + \omega^k)$, and the induction hypothesis on G_k implies that such a path exists.

(2) If there is no path from C to D in G_k, the argument reduces to the preceding claim as in the G_0 case.

If there is a path from C to D in G_k, we again use induction on the length ℓ of the path to show that every configuration on the path occurs cofinally in $[\alpha, \alpha + \omega^{k+1})$. This holds trivially when $\ell = 0$. for larger ℓ, let B be the last configuration on the path before D. From the induction hypothesis on ℓ we have that B occurs cofinally in $[\alpha, \alpha + \omega^{k+1})$. Let $m \leq k$ be the smallest value such that a $B \rightarrow D$ edge appears in G_m.

If $m = 0$, then there is a protocol transition from B to D, and fairness gives that D occurs cofinally in $[\alpha, \alpha + \omega^{k+1})$, since it is enabled in every successor to a time at which B occurs. For larger m, we have from the definition of G_m that $B \sim D$, and B appears in a terminal SCC in G_{m-1}. Now apply part (3) to show that B occurs cofinally in any interval $[\gamma, \gamma + \omega^{m-1}]$ such that B occurs at time γ. Each such interval enables D at time $\gamma + \omega^m$, and because the set of times γ is cofinal in $[\alpha, \alpha + \omega^{k+1})$, so is the set of times $\gamma + \omega^m$ at which D is enabled. So fairness says D occurs cofinally in $[\alpha, \alpha + \omega^{k+1})$.

(3) If the SCC S of G_k containing D is not terminal, then there is some path from D to a configuration B not in S, so from (1) we reach B and can't return.

Alternatively, if S is terminal, then some state $B \in S$ occurs in infinitely many intervals $[\alpha + \omega^k \cdot \ell, \alpha + \omega^k \cdot (\ell + 1))$. It is not necessarily the case that B is in a terminal SCC in G_{k-1}, but there is some configuration reachable from B that is, and since there are only finitely many possible configurations, one of these configurations A must be cofinal in $[\alpha + \omega^k \cdot \ell, \alpha + \omega^k \cdot (\ell + 1))$. That configuration A is in the same SCC S in G_k as D, because there is a path to it from D in G_k and S is terminal. So for each interval $[\alpha + \omega^k \cdot \ell, \alpha + \omega^k \cdot (\ell + 1))$ in which A occurs, D is enabled at $\alpha + \omega^k \cdot (\ell + 1)$, and since these times occur cofinally in $[\alpha, \alpha + \omega^{k+1})$, by fairness, so does D. □

Transition graphs give a translation between transfinite executions and the finite executions they represent. In this translation, transition along edges not in G_0 correspond to clock interrupts that cut off a looping computation and advance it to some future limit configuration. Each "phantom" sequence of transitions that is omitted when this occurs in effect acts as a certification that when a clock signal of a given level arrives, the current configuration could have recurred forever. By recognizing terminal SCCs directly, the transition graph approach gives an alternative certification that can be implemented in finite time on a conventional computer. We can also imagine implementing these clock ticks in practice by delivering a tick from some external source after a sufficiently long time interval, on the assumption that any reasonable physical implementation of a population protocol would converge with high probability in a bounded number of transitions.

3.1 Computation of transition graphs

Because population protocols preserve the number of agents, each graph G_k is made up of non-communicating subgraphs G_k^n for each population size n. Assuming the interaction graph of a population is complete (so that we can represent configurations with the same population counts as a single vertex), we can compute these subgraphs (and their limits!) efficiently, and use them to answer questions about population protocols running on particular inputs or input sizes.

THEOREM 3.2. *For any population protocol with m agent states and a complete interaction graph, the transition graphs $G_0^n, \ldots G_{n^{4m}}^n = G^n$ can be computed in time polynomial in n.*

PROOF. Given m different agent states, there are fewer than n^{2m} vertices in each G_k^n, where the 2 comes from the clock bit. For fixed m, this is polynomial in n.

Computing the edges in G_0^n can easily be done in polynomial time by examining the transition function. Computing the added edges in G_{k+1}^n requires finding the terminal SCCs of G_k^n, a linear-time operation using standard graph algorithms. Since $G_{k+1}^n = G_k^n$ implies $G_\ell^n = G_k^n$ for all $\ell \geq k$, we must add at least one edge to G_{k+1}^n at each step to keep going; after fewer than n^{4m} steps we will have either added all possible edges or reached the limit; in either case, $G_{n^{4m}}^n = G$. □

It follows that, in polynomial time, we can answer any questions about reachability, output-stability, and so forth that can be determined by applying standard graph algorithms to G^n. In particular, given an output-stable clocked population protocol that decides some language L, we can compute G^n, and for any input configuration determine the output in any terminal SCC reachable from some initial configuration C. This puts computation by clocked population protocols firmly in **P**, even if we allow unbounded transfinite time (in real life, arbitrarily deep layers of clocks) for them to complete.

But we can say more than this. The same argument that shows that computing G_k^n can be done in time polynomial in n also shows:

THEOREM 3.3. *For any population protocol with a complete interaction graph and any fixed k, whether there is a path from C to D in the transition graph G_k^n can be computed in nondeterministic $O(\log n)$ space.*

PROOF. Because of the nondeterminism, this problem reduces to testing if there is an edge from C to D in G_k^n. Internally, we represent C and D as a list of agent counts (thus taking $O(\log n)$ space to store each), although a logspace machine is smart enough to convert a vector of agents states into such a list and store the result if needed.

For $k = 0$, this just involves checking if the counts of each state in C and D differ by an amount that is consistent with some transition in δ, which is easily done in space $O(\log n)$.

For larger k, we make heavy use of the Immerman-Szelepcsényi Theorem [12, 17], which says that **NL** = **coNL** and more generally implies that $\mathbf{NL^{NL}} = \mathbf{coNL^{NL}} = \mathbf{NL}$ [12]. This means that as long as we recurse only to bounded depth, we can use **NL** subroutines in a **NL** or **coNL** computation and stay in **NL**.

In particular, to determine if $C \rightarrow D$ appears in G_k^n, we must check (a) if it is already in G_{k-1}^n, and if not, check (b) if $C \sim D$ and C is in a terminal SCC of G_k^n. Testing if $C \sim D$ is trivial. Testing if

C is in a terminal SCC requires testing for all B such that there is a path from C to B in G_{k-1}^n if there is also a path from B to C in G_{k-1}^n.

Fortunately, checking if there is a path from C to B in G_{k-1}^n is easily done in $\mathbf{NL^{NL}} = \mathbf{NL}$ (guess each step of the path nondeterministically, and call the $\mathbf{NL}$ oracle for edges in G_{k-1}^n to verify that each step is an edge), and similarly testing for the non-existence of a path from B to C is easily done in $\mathbf{coNL^{NL}} = \mathbf{NL}$. This leaves the universal quantifier over B, which puts the problem of detecting if C is in a terminal SCC in $\mathbf{coNL^{NL}}$, which is again equal to $\mathbf{NL}$. □

Just as Theorem 3.2 puts all languages computed by clocked population protcols in $\mathbf{P}$, Theorem 3.3 puts all languages computed by clocked population protocols using at most ω^k time in $\mathbf{NL}$. In §4.2, we will use this to show that for any fixed k, a clocked population protocol that runs in time ω^k can be replaced by a clocked population protocol that runs in time ω^2: one level of clock ticks is always enough.

4 PROGRAMMING A CLOCKED POPULATION PROTOCOL

Clocked population protocols are more powerful than standard population protocols. In this section, we give some applications of this extra power.

4.1 Leader election

Standard population protocols support leader election from an initially uniform state by fratricide: initially, each agent starts in a **leader** state L, and the transition rule $(L, L) \to (L, F)$ eventually turns all but one leader into a **follower**. But the winning leader cannot detect that this condition has occurred, and previous protocols that have used leader election as an initial stage have had to include a mechanism for restarting the computation after each remaining candidate is eliminated.

With a clock, the eventual leader can detect that it has won, because it can only receive a clock tick in a terminal SCC of the transition graph, and the only terminal SCCs are the one-leader configurations. But we can do even better than this. Fischer and Jiang [9] demonstrated that a standard population protocol cannot elect a leader starting from an arbitrary initial state, and proposed an Ω? oracle that eventually signals when there is no leader in the current configuration. We can get the same effect by using clock ticks.

Consider a protocol with three states: **leader** L, **candidate** C, and **follower** F. We use the following transition table:

$$
\begin{aligned}
L, L &\to L, F \\
L, C &\to L, F \\
C, C &\to C, F \\
F', x &\to C, x \\
C', x &\to L, x
\end{aligned}
$$

As usual, we assume that the initial configuration does not have any clock bits set. But we place no other restriction on the states of the agents. Note that we are using the convention defined earlier that F' (for example) is the state F with the clock bit set.

THEOREM 4.1. *Starting from any initial configuration with no clock bits set, the above protocol produces in time bounded by ω^2 an agent in state L'. Furthermore, if an agent reaches state L' in the above protocol, it is the only agent in the population in state L' or L.*

PROOF. Let G_k be the transition graphs for this protocol from §3. Examining the transition relation shows that terminal SCCs in G_0 come in three varieties (up to symmetry): a configuration C_F consisting of all followers, and configuration C_C containing a single candidate and the rest followers, and a configuration C_L containing a single leader and the rest followers. In each case, no agent in these stable configurations has a clock bit set.

If we reach C_F, then a configuration with some F' is enabled at time ω and at each multiple of ω thereafter until F' appears. By fairness, eventually there is some $\omega \cdot \ell$ at which at least one F' does so. The next transition involving this agent falls produces a configuration with at least one candidate, and the rest of the agents followers. The terminal SCC reachable from such a configuration consists of C_C.

If we reach C_C, then a configuration with C' is again enabled at each following multiple of ω until it occurs. When it does so, the next transition produces a single leader, and we reach the stable configuration C_L after finitely many steps.

If we reach C_L, then the same argument as before shows that after some time $\omega \cdot \ell$, L' appears.

Combining these cases shows that L' appears after $\omega \cdot \ell < \omega^2$ time, where ℓ is some finite (though unbounded) quantity.

It remains to show that when L' occurs, no other agent is in state L or L'. Under the assumption that no clock bits are set in the initial configuration, L' can only occur at a limit ordinal α before which a configuration containing L or L' occurs cofinally. But since multiple leaders are eliminated after finitely many steps, and no transition (including clock-tick transitions) ever increases the number of leaders, any configuration that occurs cofinally over a non-finite interval has at most one leader. □

4.2 Computing symmetric predicates in NL

Because agents with the same state are indistinguishable, the configuration of a population protocol with a complete interaction graph can be summarized by giving counts of the numbers of agents in each state. This essentially limits the population to storing data in unary, in the form of counters, and effectively limits population protocols to space logarithmic in the size of the population. Following the general approach of the universal randomized population protocol with a leader of Anglin *et al.* [2], we can represent a counter machine in the style of Minsky [15] by using the unique leader L remaining from the preceding construction as a finite-state controller, and expressing counter values up to n as the sum of bits scattered across all n agents. But where Anglin *et al.* were limited by the need to build a randomized **phase clock** internally out of the agents themselves, by using an external clock we can eliminate the possibility of failure and compute any symmetric predicate in $\mathbf{NL}$ on the initial agent states.

Minsky's construction shows that counter machines can simulate Turing machines, but it requires that the finite-state controller be able to test a counter for zero in addition to incrementing or decrementing it. Incrementing a counter that is not already at its maximum value consists of executing the transition $(L_1, 0) \to (L_2, 1)$ (where we omit extraneous parts of the states from the description), while decrementing a counter consists of execution $(L_1, 1) \to (L_2, 0)$. In both cases, if there is not agent with the appropriate value 0 or 1, the operation never happens.

This is a problem for standard population protocols, because they have no mechanism to detect when they have stalled. But it is not a problem with clocks: a protocol that converges to a fixed configuration will remain in that configuration at the next clock tick, and the fairness condition enforces that a clock tick is eventually delivered to the leader. So the leader can test for 0 by attempting to decrement a counter and waiting until it either succeeds or fails. (If it succeeds, an increment will restore the previous state, allowing non-destructive tests.)

A transition relation that uses clock ticks to support a decrement operation that moves the leader from state L_1 to L_2 if successful and to L_3 can be as simple as this:

$$L_1, 1 \to L_2, 0$$
$$L_1', x \to L_3, x$$

(Here x represents an arbitrary state, and transitions involving pairs not described are assumed to have no effect.)

Each such counter can count up to n, where increments and decrements that do not hit the bounds take $< \omega$ time and those that hit the bounds take $< \omega^2$ time. By using multiple counters as separate digits, we can represent values up to n^c for any fixed c. The costs of increment and decrement operations may now be only $< \omega^2$ given the need to detect overflow or underflow on lower-order digits.

Given an n^c-bounded counter, we can use it to represent a $(c \lg n)$-bit stack for binary values, where popping from the stack is done by division by two (repeatedly decrement the counter twice if possible, and increment an auxiliary counter for each pair), and pushing to the stack is done by the reverse operation of multiplying by two and adding the new value. Two such stacks make a Turing machine tape, and the leader agent can act as the finite-state controller.

For the input tape, count the number of agents in each input state, and use a counter to represent the input tape head position. We can then pretend that the input tape is sorted (because we can only compute symmetric languages). If the TM controller asks what symbol lies under the current head position, we can sum up the counter values for each possible symbol until the total exceeds the simulated head position.

So far this gets us the symmetric languages in **L**. To get **NL**, we allow the leader to both make nondeterministic choices and to reset the computation to the beginning after reaching a rejecting state. Nondeterminism does not require modifying the transition function to be nondeterministic; instead, we can use the nondeterministic scheduling of which agent the leader meets next as a supply of nondeterminism, and observe that fairness gives that if there exists an accepting computation path, we will eventually find it. By mapping all other states that the accepting state to a rejecting output,

we get an output-stable simulation of a nondeterministic logspace machine.

Careful accounting of the counter operations shows that the time per step of this machine is $< \omega^2$. Because we converge to a terminal SCC (either an accepting state or an endless loop of rejecting computation paths) in finitely many steps, the total time to reach an output-stable configuration is also $< \omega^2$. In the next section, we show that it is possible to detect when an output-stable configuration occurs.

4.3 Detecting stable outputs

The ability to simulate **NL** predicates means that we can detect when a clocked population protocol has reached an output-stable configuration, provided it does so in time $< \omega^k$ for some fixed k. This means that unlike standard population protocols, we can compose clocked population protocols sequentially, using the output of one protocol as the input to the next.

Here is the idea: At any time, the leader may nondeterministically choose to test for an output-stable configuration. To do so, the leader switches to transitions of the form $(L, x) \to (L, \hat{x})$, where $\hat{x}$ represents a "frozen" version of state x. Eventually, no more unfrozen agents remain, and the protocol enters a configuration that is stable until the next clock tick.

Upon receiving a clock tick, the leader can tell that it has successfully frozen all the agents, and proceeds to the verification step. The frozen configuration C will be output-stable if every configuration reachable from it in $< \omega^k$ time has the same uniform output on all agents. We now observe that the **NL** simulation of the preceding section allows the leader to perform an **NL** computation of its choosing on the frozen state in $< \omega^2$ time, so we need only find a way to test output-stability in **NL**. Using an **NL** machine, we can nondeterministically guess the common output x. The question then becomes, is there a sequence of transitions starting from the frozen state that gives some agent a different output $y \neq x$? The existence of such a sequence is testable in **NL** by Theorem 3.3: we nondeterministically guess a bad configuration D and then apply the theorem to check if D is reachable from C in $< \omega^k$ time. But what we want to know is that no such sequence exists. This question is in **coNL**, which puts output-stability in $\mathbf{L^{coNL}}$. But this class is equal to **NL** by the Immerman-Szelepcsényi Theorem.

This means that whenever the leader freezes the configuration, it can use the procedure of the preceding section to try one branch of the **NL** computation that tests output-stability (note that this will require additional state in the agents separate from the frozen states). If the configuration is in fact output-stable and the protocol picks the right branch, we are done. If not, the leader reverses the freezing process and restarts the underlying protocol.

4.4 Reduction to $< \omega^2$ time

We can also apply Theorem 3.3 to not only detect convergence of a population protocol that runs in $< \omega^k$ time but to reduce its cost to $< \omega^2$ time. Given a protocol that runs in $< \omega^k$ time, we know that it computes a symmetric function in **NL**. We can thus ignore the original protocol and simulate the protocol using the method of §4.2 to compute the same predicate, output-stably, in time $< \omega^2$. Adding the convergence detector of §4.3 gives us the

ability to detect when the output has stabilized: if we fire it only after clock ticks, we can guarantee that it runs only finitely many times using $< \omega^2$ time each, for a total cost that is still $< \omega^2$. So we can in fact compute any predicate that is stably computable by a clocked population protocol that runs in $< \omega^k$ time for any fixed k in $< \omega^2$ time.

We summarize this result as:

THEOREM 4.2. *Any predicate computable by a clocked population protocol in $< \omega^k$ time, for fixed k, is computable in $< \omega^2$ time.*

5 RELATION TO OTHER ORACLES

The clock ticks added in the clocked population protocol model are a kind of oracle, providing the protocol with information ("you are stuck") that it cannot obtain for itself. We have seen that for the purposes of leader election, clock ticks are at least as powerful as the Ω? oracle of Fischer and Jiang. A natural question is whether other kinds of oracles can be simulated using clock ticks or vice versa.

5.1 Higher-order clock ticks

We think of the multiples of ω as the smallest clock interval in the system, while multiples of larger ordinals like ω^2 represent longer intervals. The clock mechanism by itself does not allow the agents to distinguish between a tick that arrives at a multiple of ω that is not a multiple of ω^2 from one that arrives at a multiple of ω^2. An obvious extension is to add $2, 3, \ldots, k$ values to the existing 0 and 1 clock values so that a clock tick i is delivered only at times that are multiple of ω^i.

But this is not necessary. Having elected a unique leader, we can use it together with a counter implemented across the other agents to simulate such higher-order clock ticks for multiples of ω^k up to the maximum value of the counter.

Let x represent the value in the counter. Initially, x is 0. When the leader receives a clock tick, it increments x and then resets it to zero. (As part of the reset operation, it may also copy the counter value somewhere else for later use.)

THEOREM 5.1. *Using the above mechanism, the leader increments the counter to k following a clock tick only at times that are multiples of ω^k.*

PROOF. By induction on k. The base case is $k = 0$; all limit ordinals are multiples of $\omega^{0+1} = \omega$, so the hypothesis holds trivially.

For larger k, observe that $x = k$ can occur at α if and only if configurations with $x = k$ are cofinal in α. Each such configuration must occur at a time that is a multiple of ω^k. Expand α as $\omega^{k+1} \cdot \beta + \omega^k \cdot q + \rho$, where q is finite and $\rho < \omega^k$. If either γ or ρ is nonzero, there there exists a time ν less than α such that no multiples of ω^k occur after ν. But then configurations with $x = k$ are not cofinal in α. It follows that $q = \rho = 0$ and α is a multiple of ω^{k+1}. □

Note that these higher-order clock ticks are not the strongest we can imagine: for example, when the leader sees $x = 2$, it may be that the current time is ω^3 or some even larger power of ω. It is not clear whether providing an exact measure of the exponent would give a protocol more power directly, although if k is bounded by a constant, we can extend the transition graph construction from §3

to enforce this constraint within the transition graph model, and compute the same predicates as a clocked population-protocol with these stronger higher-order ticks using the same approach as in Theorem 4.2.

5.2 Absence detectors

The **absence detector** of Michail and Spirakis [14] allows agents to determine precisely which states are present in the population at the time of encountering the detector. This allows construction of counters as in §4.2, and generally allows computation of symmetric predicates in **NL**.

Clock ticks are weaker than absence detectors in the sense that delivery of a clock tick can only indicate that no progress can be made. This can be used to test for the absence of a particular state q by a leader that changes state if it encounters a q, but only if no other activity is ongoing in the population. This suggests simulating an absence detector directly by applying the same freezing-and-unfreezing method used in §4.3, but the cost of triggering this mechanism routinely during the execution of a larger protocol might be high.

On the other hand, it is straightforward to modify the transition graph construction of §3 to include an absence detector, and an **NL** machine that has access to the entire population count can easily implement an absence detector while simulating a population protocol. So the fact that clocked population protocols can compute functions in **NL** gives them the same computational power as a population protocol with an absence detector.

6 CONCLUSIONS

We have shown that allowing an external clock to deliver ticks to a stuck population protocol extends its power dramatically, giving it the ability to compute any symmetric predicate in **NL**. This generalizes previous results [1, 14, 16] that obtained similar power using more specialized mechanisms to detect convergence.

By using a model of transfinite executions over time intervals represented using ordinal arithmetic, we showed that the transition rules and fairness condition of the standard model extend in a straightforward way to these clocked population protocols. We also gave a representation of these transfinite executions using paths over finite transition graphs, demonstrating that this model both corresponds to a clock mechanism that could be implemented in practice in finite time, that the added power still remains plausibly within **P**, and that clocked population protocols have a capability for introspection, allowing clocked population protocols to compute properties of the executions of other clocked population protocols and related models by applying **NL** computations to their transition graphs.

Our results apply to population protocols with complete interaction graphs, the weakest form of the standard model. An interesting question is how clock bits and their representation in terms of transfinite fair executions would interact with population protocols with less symmetric interaction graphs, or even with other distributed computing models providing by their own eventual progress guarantees.

REFERENCES

[1] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. 2006. Computation in networks of passively mobile finite-state sensors. *Distributed Computing* (March 2006), 235–253.

[2] Dana Angluin, James Aspnes, and David Eisenstat. 2008. Fast computation by population protocols with a leader. *Distributed Computing* 21, 3 (Sept. 2008), 183–199.

[3] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. 2007. The computational power of population protocols. *Distributed Computing* 20, 4 (Nov. 2007), 279–304.

[4] Joffroy Beauquier, Peva Blanchard, and Janna Burman. 2013. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *International Conference On Principles Of Distributed Systems*. Springer, 38–52.

[5] Joffroy Beauquier, Peva Blanchard, Janna Burman, and Oksana Denysyuk. 2016. On the Power of Oracle Ω? for Self-Stabilizing Leader Election in Population Protocols. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Springer, 20–35.

[6] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G Spirakis. 2010. All symmetric predicates in NSPACE (n 2) are stably computable by the mediated population protocol model. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 270–281.

[7] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. 2011. Passively mobile communicating machines that use restricted space. *Theor. Comput. Sci.* 412, 46 (2011), 6469–6483. https://doi.org/10.1016/j.tcs.2011.07.001

[8] Ioannis Chatzigiannakis, Othon Michail, and Paul G Spirakis. 2009. Mediated population protocols. In *International Colloquium on Automata, Languages, and Programming*. Springer, 363–374.

[9] Michael Fischer and Hong Jiang. 2006. Self-stabilizing leader election in networks of finite-state anonymous agents. In *International Conference On Principles Of Distributed Systems*. Springer, 395–409.

[10] Rachid Guerraoui and Eric Ruppert. 2009. Names trump malice: Tiny mobile agents can tolerate byzantine failures. In *International Colloquium on Automata, Languages, and Programming*. Springer, 484–495.

[11] Joel David Hamkins and Andy Lewis. 2000. Infinite time Turing machines. *The Journal of Symbolic Logic* 65, 02 (2000), 567–604.

[12] Neil Immerman. 1988. Nondeterministic Space is Closed Under Complementation. *SIAM J. Comput.* 17, 5 (1988), 935–938. https://doi.org/10.1137/0217058

[13] Thomas Jech. 2002. *Set Theory: The Third Millenium Edition: revised and expanded*. Springer.

[14] Othon Michail and Paul G Spirakis. 2015. Terminating population protocols via some minimal global knowledge assumptions. *J. Parallel and Distrib. Comput.* 81 (2015), 1–10.

[15] Marvin L. Minsky. 1961. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics* 74, 3 (Nov. 1961), 437–455.

[16] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. 2008. Computation with finite stochastic chemical reaction networks. *natural computing* 7, 4 (2008), 615–633.

[17] Róbert Szelepcsényi. 1988. The method of forced enumeration for nondeterministic automata. *Acta Informatica* 26, 3 (1988), 279–284. https://doi.org/10.1007/BF00299636

[18] James F. Thomson. 1954. Tasks and super-tasks. *Analysis* 15, 1 (1954), 1–13.

[19] Johann v. Neumann. 1923. Zur Einführung der transfiniten Zahlen. *Acta Universitatis Szegediensis* 1 (1923), 199–208.