Clocked Population Protocols[*]

James Aspnes[†]

May 12, 2021

Abstract

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 for the protocol to have converged. We represent "long enough" as an infinite time interval, and treat the protocol as operating over transfinite time. Over finite time intervals, the protocol behaves as in the standard model. At nonzero limit ordinals, corresponding to clock ticks, the protocol switches to a limit of previous configurations supplemented by a clock signal appearing as an extra component in some of the agents' states. Fairness generalizes to transfinite executions straightforwardly. 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 population size, and can be simulated using only finitely many clock ticks.

1 Introduction

A **population protocol** [AAD$^+$06] 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 [AAE08, SCWB08]. 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 [AAER07],

[*]An earlier version of this paper appeared as [Asp17].

[†]Yale University, Department of Computer Science. Supported in part by NSF grants CCF-1637385 and CCF-1650596.

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 [MS15]. 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 (Theorem 4.1), and vice versa (Corollary 3.4).

An advantage of showing that clocked population protocols have power equivalent to $\mathbf{NL}$ is that we can use closure results on $\mathbf{NL}$ to program clocked population protocols. For example, the Immerman-Szelepcsényi

Theorem [Imm88, Sze88] 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 the graph of configurations reachable without any agent receiving a clock tick. We can thus think of the transfinite-execution model as the limit behavior of finite-execution protocols in which the clock ticks are delivered after a long enough interval for this event to occur. Alternatively, the characterization in terms of terminal strongly-connected components gives an algorithmic method for determining when to send a clock tick, showing that the result of an output-stable clocked population protocols can be simulated in polynomial time.

We show in particular 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 [GR09], 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 $\mathbf{NSPACE}(n \log n)$. The still stronger **mediated population protocol** model of Michail *et al.* [MCS11], which stores information on edges between agents, boosts this power still further to $\mathbf{NSPACE}(n^2)$ [CMN+10]. It is also possible to expand the space in each agent, which is $O(1)$ in the standard model. Chatzigiannakis *et al.* [CMN+11] have shown that even a modest extension to $\Theta(\log n)$ bits per agent allows computations of symmetric predicates in $\mathbf{NSPACE}(n \log n)$ (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.

The clocked population protocol is an attempt to generalize previous work on absence detectors and cover-time oracles to allow a protocol to detect convergence. An alternative approach, recently suggested by Blondin *et al.* [BEJ19], is to add a reliable broadcast mechanism. They show that this also gives a population power the ability to compute symmetric predicates in **NL**, and may have a more practical implementation in models where convergence times are not predictable enough to rely on waiting.

The idea of modeling computation over transfinite time has precedent in work on transfinite Turing machines [HL00] as a model for **supertasks** [Tho54], 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 [Jec02].

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 [vN23] 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 $\{0, 1, 2, \ldots; \omega, \omega + 1\}$, 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 $\{0, 1, 2, \ldots; \omega, \omega + 1, \omega + 2, \ldots\}$. 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 $\rho < \beta$ [Jec02, Lemma 2.25(iv)].

Cantor's Normal Form Theorem (see [Jec02, Theorem 2.26]) says that any ordinal α has a unique representation $\alpha = \omega^{\beta_1} \cdot k_1 + \dots + \dots \omega^{\beta_n} \cdot k_n$, where n is finite, $\alpha \geq \beta_1 > \beta_2 > \dots > \beta_n$ 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.

[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.

In the standard model [AAD+06], 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 $O : Q \to Y$ 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 by 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.

An **initial configuration** is obtained from an input word by establishing a bijection between positions in the input word and agents in the population, and applying the input function I to map the symbol in each position to the initial state of the corresponding agent. Note that because populations are generally unstructured, this may lose information about the order of the symbols in the input.

A configuration C is **output-stable** if any configuration reachable from C has the same output as C, where the usual convention is that the output is a common output value found at every agent by applying O to the agent's state. A population protocol computes a predicate on input configurations if every fair execution eventually reaches an output-stable configuration with the correct output. To enable comparison with more traditional computational models, we also adopt the usual convention of saying that a population protocol decides a language L if it is output-stable and the set of strings in L are precisely those that map to input configurations for which the output of the language is true.

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 both the input function I and the transition function δ always produce states in which the clock bits are set to 0.

This gives δ the type $Q \times \{0,1\} \times Q \times \{0,1\} \to Q \times Q$, where the extra bits in the input 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 [FJ06]. However, changes to the clock bits are more restricted than in this and subsequent work applying oracles to population protocols (e.g., [BBB13, BBBD16]). 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. For executions indexed by the natural numbers, this corresponds to reaching a configuration that occurs infinitely often in fair executions. With a transfinite timescale, this generalizes to configurations that occur at unboundedly large times over an interval, a property known as **cofinality**. We allow a configuration to appear at a limit time if it occurs at arbitrarily large times up to that time, and allow clock bits to be set in this limit configuration arbitrarily. This is combined with an extended fairness condition, which also generalizes configurations occurring infinitely often to configurations occurring at arbitrary large times within some infinite interval. The extended fairness condition enforces that clock bits are eventually set, effectively allowing the protocol the ability to identify its own limit behavior when it observes them. (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 $2, 4, 8, 16, \dots$ 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.

For any ordinal α, an **execution** Ξ of length α consists of a sequence of configurations C_β, indexed by the ordinals $\beta < \alpha$, that satisfies certain consistency constraints that we now define. A configuration C is **enabled** following a sequence Ξ of configurations 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 a sequence of configurations Ξ such that each configuration C_β is enabled following the prefix of Ξ of length β.

We will refer to transitions between a configuration C_α and its successor configuration $C_{\alpha+1}$ as an **ordinary transition**. For symmetry, when α is a nonzero limit ordinal, we will refer to C_α as being produced by a **limit transition**, even though in this case the "transition" depends on the entire sequence of configurations leading up to α and not any single configuration.

Because the transition function δ does not output clock bits, an interaction between two agents in an ordinary transition resets any clock bits that they might be carrying. The only way to set clock bits is at a limit transition, where allowing any configuration that is equivalent to a cofinally-occurring configuration means that any combination of clock bits (or none) is possible. We will need the fairness condition (defined in §2.4) to enforce that in any fair execution, some clock bits are eventually set.

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 nonzero limit ordinals captures the intuition of waiting long enough to converge to some subset of configurations, at least for ordinals of the form $\alpha + \omega$, because the only configurations enabled at time $\alpha + \omega$ are those that appear infinitely often since the last clock tick 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 will we do below in §2.4 by extending the standard global fairness condition to transfinite intervals. But first we give an example of a simple clocked population protocol to illustrate how clocked executions work.

2.3 Example: leader election

We will use leader election to illustrate application of the clock bits.

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 [FJ06] 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 initial states of the agents. For example, it is possible that every agent starts in state F. Note that we are using the convention defined earlier that F' (for example) is the state F with the clock bit set.

What we would like this protocol to do is proceed through the following sequence of intermediate configurations (some of which may be skipped depending on the initial configuration):

1. Starting from an all-F configuration, eventually at least one F receives a clock tick and promotes itself to C.

2. Starting from a configuration with at least one C agent and no L agents, eventually it reaches a configuration with only one C agent and the rest F agents.

3. Starting from a configuration with one C and the rest F, eventually the C receives a clock tick which eventually promotes it to L.

4. Starting from a configuration with at least one L agent, eventually it reaches a configuration with exactly one L agent and the rest F.

5. Starting from a configuration with one L and the rest F, eventually the L receives a clock tick.

It is not hard to show that each of these steps *can* occur. For (1), since the all-F configuration is inescapable through ordinary transitions, so it will occur cofinally up to the next limit ordinal. So we can deliver a tick to some F at the next limit ordinal step, then apply the $F', x \to C, x$ rule to generate a candidate. For (2), we can just apply the rule $C, C \to C, F$ until only one C remains. Similar arguments apply to the remaining steps. But we cannot yet show that each step *must* occur. Doing so requires extending the fairness condition.

2.4 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 the set of times at which C occurs is also cofinal in α. We define an execution prefix to be fair if the same condition holds for any α less than or equal to the length of the execution prefix.

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 $C_1 \to C_2$, 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 $C_1 \to C_2$, and C_1 occurs infinitely often in some interval $[\alpha, \alpha + \omega)$, C_2 also occurs infinitely often in $[\alpha, \alpha + \omega)$.*

Proof. Let $\alpha + k_1 < \alpha + k_2 < \dots$ enumerate the times at which C_1 appears in $[\alpha, \alpha + \omega)$. Because the sequence $k_1, k_2, \dots$ is infinite, it is unbounded in ω, thus $\alpha + k_1, \alpha + k_2, \dots$ 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.

It is worth noting that this definition of fairness respects concatenation of executions:

Lemma 2.2. *Let Ξ_1, Ξ_2, $\ldots \Xi_k$ be a finite sequence of fair execution prefixes such that the initial configuration in Ξ_{i+1} is enabled following Ξ_i. Then the concatenation $\Xi_1 \Xi_2 \ldots \Xi_k$ is a fair execution.*

Proof. For each i, let α_i be the length of Ξ_i, and let $\alpha = \sum_{i=1}^{k} \alpha_i$ be the total length of the combined execution. If $\beta \leq \alpha$, there exists some minimum j such that $\beta \leq \sum_{i=1}^{j} \alpha_j$. Now if C is enabled cofinally in β, then C is enabled cofinally in some prefix of Ξ_j of length γ, where $\beta = \sum_{i=1}^{j-1} \alpha_i + \gamma$. But then fairness of Ξ_j says that C occurs cofinally in γ, which means that it occurs cofinally in β. ☐

2.5 Leader election in fair executions

We can now justify the claim that the example protocol from §2.3 does in fact elect a leader.

Theorem 2.3. *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 the protocol reaches a configuration with an agent in state L', this will be the only agent in the configuration in state L' or L, with all other agents in state F' or F.*

Proof. The proof follows the informal argument from §2.3. We will make heavy use of Lemma 2.1 to show convergence to desirable configurations within intervals of length ω, then apply the fairness condition to longer intervals to show that clock ticks eventually allow further progress.

Let C_α be the configuration at time α, where α is an arbitrary ordinal bounded by ω^2. There are several possible cases:

1. C_α is an all-F configuration. Because all transitions are no-ops, C_α is cofinal in the interval $[\alpha, \alpha + \omega)$. It follows that a configuration in which at least one agent is in state F' is enabled at time $\alpha + \omega$; if this configuration occurs as $C_{\alpha+\omega}$, then (applying Lemma 2.1) either (a) at least one such F' agent participates in an $F', x \to C, x$ transition in the interval $[\alpha + \omega, \alpha + \omega \cdot 2)$, or (b) all clock ticks are lost during this interval, leaving the protocol again in the all-F configuration.

   If we get a C, fall through to the next case. Otherwise, repeat the same argument starting at the first configuration $C_{\alpha+\omega+k}$ that is all-F. Iterating this process eventually either generates a C, or it produces an execution over $[\alpha, \alpha + \omega^2)$ in which some configuration $C_{F'}$ with at least one F' is enabled at every time $\alpha + \omega \cdot k$ (where $k \in \mathbb{N}$), meaning that $C_{F'}$ is enabled cofinally in $\alpha + \omega^2$. But then the fairness condition requires that $C_{F'}$ occurs cofinally in $\alpha + \omega^2$. For each time $\alpha + \omega \cdot k + \ell$ at which $C_{F'}$ occurs, a configuration C_C with at least one C is enabled at time $\alpha + \omega \cdot k + \ell + 1$. This sequence of times is also cofinal in $\alpha + \omega^2$, so C_C must occur cofinally (and thus at least once) in $\alpha + w^2$. In particular there is some first time $\alpha' < \alpha + \omega^2 = \omega^2$ at which C_C occurs. So we can always fall through to the next case (with a new value of α).

2. C_α contains at least one C, with all other agents F. Suppose C_α contains $m > 1$ copies of C. Then by Lemma 2.1, if C_α occurs infinitely often in $[\alpha, \alpha+\omega)$, so does some configuration C' with only $m-1$ copies of C. But there are no transitions that can increase the number of C agents, so this in fact implies that a configuration with $m-1$ C agents is reached after some finite interval $[\alpha, \alpha+k)$. Repeating this argument yields a configuration $C_{\alpha+k'}$ with exactly one C (and all other agents F) at some time $\alpha + k'$, where k' is finite. No transitions will change this configuration, making $C_{\alpha+k'}$ cofinal in $[\alpha, \alpha + \omega)$. This enables a configuration where the unique C agent gets a clock bit at time $\alpha + \omega$.

   As in the previous case, we can argue that such a C'-containing configuration is in fact enabled at every time $\alpha + \omega \cdot k$ for which a configuration with at least one C occurs in $[\alpha + \omega \cdot (k-1), \alpha + \omega \cdot k)$. From a C'-containing configuration, we can transition to an L-containing configuration via the $C', x \to L, x$ rule. If this transition occurs, we reach the next case, where there is one L agent and the rest F. If not, C'-containing configurations are enabled cofinally in $[\alpha, \alpha + \omega^2)$, implying that C'-containing configurations occur cofinally in $[\alpha, \alpha + \omega^2)$; but then L-containing configurations are also enabled cofinally in $[\alpha, \alpha + \omega^2)$, which means that some such configuration eventually occurs. Either way, we fall through to the next case.

3. C_α contains at least one L and the rest of the agents C or F. From Lemma 2.1, at some point $\alpha + k$ in $[\alpha, \alpha + \omega)$ we reach a configuration where one L consumes all other L and C agents. Then for each time $\alpha + \omega \cdot k$, where $0 < k < \omega$, a configuration in which this L agent gets a clock tick is enabled. This sequence of times is cofinal in $[\alpha, \alpha + \omega^2)$, so eventually one of these configurations is reached.

Each of the above cases takes a time $\beta_i < \omega^2$. Writing each time as $\beta_i = \omega \cdot k_i$, where k_i is finite, we get a total time of $\omega \cdot (k_1 + k_2 + k + 3) < \omega^2$. This shows that we reach a configuration with at least one L' in strictly less than ω^2 time.

For the second part, observe that we can generate L' only in a configuration indexed by a nonzero limit ordinal $\alpha < \omega^2$, which must be of the form $\omega \cdot k$ for $k > 0$. For this to occur, a configuration containing at least one L must be cofinal in $[\omega \cdot (k-1), \omega \cdot k)$. But then Lemma 2.1 implies that all other L or C agents are removed in this interval, so that only configurations with at most one L and no C's are cofinal in $[\omega \cdot (k-1), \omega \cdot k)$. So the L' appearing at time $\omega \cdot k$ is unique, and all other agents must be in state F or F'. □

2.6 Output-stability and predicate computation

The notion of output-stable configurations carries over directly to transfinite executions: an output-stable configuration C has the property that no configuration C' that occurs later within the same execution has a different output. A clocked population protocol computes a predicate if it eventually reaches an output-stable configuration with the correct output in all executions. The time at which the protocol computes the predicate in a particular execution is the first time at which it reaches such a configuration.

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 nonempty 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 C_1 in G_k might occur cofinally in an interval $[\alpha, \alpha + \omega^k)$ in a fair execution.

At a minimum, we need $C_1 \rightarrow_k^* C_1$, so that C_1 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_1. This essentially means that no such state C_2 can be reachable from C_1 at all, or in other words that C_1 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 $E_{k+1} = E_k \cup \{(C_1, C_2) \mid C_1 \sim C_2 \text{ and } C_1 \text{ is in a terminal SCC in } G_k\}$. We will refer to the edges in $E_{k+1} \setminus E_k$ as level $k+1$ edges.

The full transition graph G_ω 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_ω will in general be a limit $\bigcup_{k=0}^{\infty} G_k$. Fortunately, we do not have to go past ω to compute G_ω, 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 simultaneous 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 $\alpha \leq \beta < \alpha + \omega^{k+1} = \alpha + \omega$. 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 $d(C, D)$, the length of the shortest path from C to D in G_0. 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.

   Rewrite β as $\alpha + \beta'$, where $\beta' < \omega^{k+1}$, and use ordinal division to rewrite $\beta' = \omega^k \cdot q + \rho$, where the quotient q is finite (because otherwise $\beta' \geq \omega^k \cdot q \geq \omega^k \cdot \omega = \omega^{k+1}$) and the remainder $\rho < \omega^k$. This gives an expansion of β as $\alpha + \omega^k \cdot q + \rho$, where q is finite and $\rho < \omega^k$. 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 part (3) of 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 $B \to D$ 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, transitions 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 disconnected 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 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 $G_0^n, \dots G_{n^{4m}}^n = G_\omega^n$ 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, $G_{n^{4m}}^n = G_\omega^n$. □

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_ω^n. In particular, given an output-stable clocked population protocol, we can compute G_ω^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 practical terms, arbitrarily deep layers of clocks) for them to complete.

But we can say more than this. The same argument that shows that computing G^n_k 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^n_k 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^n_k. Internally, we represent C and D as a list of agent counts, which takes $O(\log n)$ space to store each.

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 [Imm88, Sze88], which says that $\mathbf{NL} = \mathbf{coNL}$ and more generally implies that $\mathbf{NL}^{\mathbf{NL}} = \mathbf{coNL}^{\mathbf{NL}} = \mathbf{NL}$ [Imm88]. 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 $C \to D$ appears in G^n_k, we must check (a) if it is already in G^n_{k-1}, and if not, check (b) if $C \sim D$ and C is in a terminal SCC of G^n_k. Testing if $C \sim D$ is trivial. Testing if C is in a terminal SCC requires testing for all B for which there is a path from C to B in G^n_{k-1}, if there is also a path from B to C in G^n_{k-1}.

Fortunately, checking if there is a path from C to B in G^n_{k-1} is easily done in $\mathbf{NL}^{\mathbf{NL}} = \mathbf{NL}$ (guess each step of the path nondeterministically, and call the **NL** oracle for edges in G^n_{k-1} 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 $\mathbf{coNL}^{\mathbf{NL}} = \mathbf{NL}$. This leaves the universal quantifier over B, which puts the problem of detecting if C is in a terminal SCC in $\mathbf{coNL}^{\mathbf{NL}}$, which is again equal to **NL**. $\square$

Just as Theorem 3.2 puts all languages computed by clocked population protocols in **P**, Theorem 4.1 puts all languages computed by clocked population protocols using at most ω^k time in **NL**. The proof is essentially the same as for **P**, as **NL** can determine whether there exists a path from the initial configruation to some (nondeterministically guessed) state in G^n_k with a particular output using Theorem 4.1, and verify that there are in fact no states with different outputs by using Theorem 4.1 again together with the closure of **NL** under complement given by the Immerman-Szelepcsényi theorem. For reference, we state this as an explicit corollary:

Corollary 3.4. *Any language computed by a clocked population protocol in at most ω^k time, for any fixed k, is in* **NL**.

In §4.1, we will apply this result 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, as we have already seen with Theorem 2.3. In this section, we give further applications of this power.

4.1 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 effectively limits the population to storing data in unary, in the form of counters, which means that a configuration of a population protocol can be written down in a more traditional computational model in space only logarithmic in the size of the population. Following the general approach of the universal randomized population protocol with a leader of Angluin *et al.* [AAE08], we can represent a counter machine in the style of Minsky [Min61] 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 Angluin *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 **NL** on the initial agent states correctly in all executions.

As this is now a standard construction in the population protocol literature (see, for example, [MS15, BEJ19]), we concentrate on the central question of implementing a counter holding values up to $O(n)$ that supports increment, decrement, and test for zero. Multiple instances of this counter stored in parallel can the then be used to implement counters of polynomial size [FMR68, Lemma 3.2], which can in turn simulate a Turing machine tape of size $O(\log n)$ [FMR68, Theorem 3.1].

We assume that we start with a leader, which can be elected using the protocol given in §2.3. Each agent stores its original input; in addition, the leader holds the finite-state control for the counter machine, while the other agents store a vector of k bits, each representing part of a unary counter.

Incrementing a counter that is not already at its maximum value consists of executing a transition $(L_1, 0) \to (L_2, 1)$, where L_1 and L_2 are different states of the leader and we omit extraneous parts of the states from the description. Decrementing a counter consists of execution $(L_1, 1) \to (L_2, 0)$. In both cases, if there is no 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 otherwise can be as simple as this:

$$\begin{gathered} L_1, 1 \to L_2, 0 \\ L_1', x \to L_3, x \end{gathered}$$

(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 (including those used in zero tests) take $< \omega^2$ time. Since any finite number of operations that take $< \omega^2$ time also finish in $< \omega^2$ time, this means that we can implement any finite counter machine execution in $< \omega^2$ time. Using the results of [FMR68], this gives the ability to compute any symmetric predicate in **L** in $< \omega^2$ time.

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 nondeterministic bits, and observe that fairness implies 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.

Since each step of this machine involves a finite number of counter-machine steps, 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 $< \omega^2$. It follows that:

Theorem 4.1. *For any symmetric predicate P in* **NL***, there is a clocked population protocol that stably computes P.*

In the next section, we show that it is possible not only to reach an output-stable configuration, but to detect when an output-stable configuration occurs.

4.2 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.

Recall that a configuration C is output-stable if there is no configuration C' reachable from C with a different output. In our model, this includes both configurations reachable by ordinary transitions and by clock transitions. We first observe that if an output-stable configuration is always reached in time $< \omega^k$, then testing output-stability for some configuration C requires only testing that C has the same output as every configuration C' that is reachable from C in time $< \omega^k$.

The reason for this is that for any configuration C'' that is eventually reachable from C (possibly at time ω^k or later), then between C and C'' must come some output-stable configuration C' at time $< \omega^k$. If all such C' have the same output as C, then output-stability of each C' implies that all such C'' also have the same output as C: so in this case, C itself is output-stable. So we just need a mechanism to test if some configuration with a different output is reachable from C in time $< \omega^k$.

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. Since it is given that the simulated protocol reaches an output-stable configuration within ω^k time, 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 4.1: 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 $\mathbf{NL}^{\mathbf{coNL}}$. 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.

4.3 Reduction to $< \omega^2$ time

We can also apply Theorem 4.1 to not only detect convergence of a population protocol that runs in $< \omega^k$ time but to reduce its time to $< \omega^2$. 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 Theorem 4.1 to compute the same predicate, output-stably, in time $< \omega^2$. Adding the convergence detector of §4.2 gives us the ability to detect when the output has stabilized: if we structure the transition relation so that the convergence detector starts only in a configuration with a clock tick, 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.*

4.4 Alternation

Alternation [CKS81] extends a complexity model by allowing branching computations that include both existential ($\exists$) branches and universal ($\forall$) branches. This replaces the usual single-threaded computation with a tree of hypothetical computations, with the value of the computation computed from the values reported at the leaves by applying OR at $\exists$ nodes and AND at $\forall$ nodes.

The branching nodes are organized into alternating layers consisting entirely of $\exists$ or $\forall$ branches. The output of an alternating computation can be expressed by a Σ^k formula of the form $\exists w_1 \forall w_2 \exists w_3 \dots Q w_k \colon \Phi(w_1, \dots, w_k, x)$ or a Π^k formula of the form $\forall w_1 \exists w_2 \forall w_3 \dots Q w_k \colon \Phi(w_1, \dots, w_k, x)$. In both cases, Q is $\exists$ or $\forall$ as appropriate, the w_i variables represent which branch the computation takes at each branching node, and the predicate Φ represents the output of the computation on input x for these particular choices.

Note that the w_i may encode more information than the computational model is capable for storing; for example, the class $\mathbf{NL} = \mathbf{\Sigma}^1_{\mathbf{L}}$ can detect s–t paths by branching over all possible paths of length at most n, while storing only the most recent nodes on the path. In effect, each w_i can be represented by a configuration C_i representing the final configuration resulting from the choice made in w_i—like the final node in the path, in the s–t connectivity example.

Population protocols, clocked or not, include inherent nondeterminism in their schedules, so in the context of a clocked population protocol, the natural interpretation of a branching node is a choice between possible next steps. Here w_i represents some sequence of nondeterministic choices made during an executing and the corresponding C_i will be the resulting configuration. So we get nondeterminism for free. The question is whether we can evaluate the sequence of alternating $\exists$ and $\forall$ quantifiers.

We can implement bounded alternation directly in a clocked population protocol using essentially the same technique used to implement $\mathbf{NL} = \mathbf{\Sigma}^1_{\mathbf{L}}$. Suppose we wish to compute a predicate of the form $\exists w_1 \colon \Phi(w_1, C_0)$, where $\Phi(w_1, C_0)$ represents the result of a clocked population protocol that runs starting in some configuration C_1 resulting from applying the schedule determined by w_1 starting in configuration C_0.

Elect a leader, and have the leader make a spare copy of C_0. Run the protocol using the copy to reach C_1, then run the remainder of the protocol, detecting termination if necessary using the technique in §4.2. If the leader sees 1 as the output of this remainder, terminate with 1 as the output; otherwise, restart from the saved C_0.

Because all output-0 configurations can reach the restart configuration and vice versa, the output-0 configurations form a strongly-connected component. This component is terminal only if there is no output-1 configuration, so if there is no output-1 configuration, the leader will eventually receive a clock tick while in an output-0 configuration. In this case the leader can decide 0.

By inverting the leader's output, we can equally well compute predicates of the form $\forall w_1 : \Phi(w_1, C_0)$. Iterating the construction gives arbitrary Σ^k and Π^k formulas, for any fixed k.

It is not clear that this gives clocked population protocols any additional power, because all of the predicates that we know how to compute using such a protocol are contained in **NL**, and $\mathbf{\Sigma}^k_{\mathbf{NL}} = \mathbf{\Pi}^k_{\mathbf{NL}} = \mathbf{NL}$ for any fixed k due to the Immerman-Szelepcsényi theorem. However, there is an intriguing possibility if we could remove the need to store a stack of k intermediate configurations: *unbounded* alternation would allow us to compute any predicate on initial configurations in **P**, since unbounded alternation over log space gives the class **AL** = **P** [CKS81]. Because we already know that any such predicates can be computed in **P** (Theorem 3.2), this would make the class of predicates computable by clocked population protocols precisely equal to the symmetric predicates in **P**. But it is not clear how to implement unbounded alternation without requiring unbounded space.[2]

[2]There are other ways clocked population protocols might turn out to have the power of **P**. One is that **NL** might in fact equal **P**, an open question at this time. Another is that it might be possible to implement the class **FO**(LFP) of first-order formulas with a **least-fixed-point** operator implicitly using the transition graphs defined in §3, which is known to be equivalent to **P** by the Immerman-Vardi Theorem [Imm86, Var82].

The idea is that a least-fixed-point operator LFP($\Phi(P, x)$) constructs a predicate from a first-order formula Φ in an unspecified predicate P, by starting with an empty predicate P_0 and defining $P_{i+1}(y) = P_i(y) \vee \Phi(P_i, y)$, stopping when $P_{i+1} = P_i$. The family of transition graphs $G_0, G_1, \ldots$ computed in the proof of Theorem 3.2 match this structure exactly. The question then is whether a particular predicate Φ needed for the Immerman-Vardi Theorem could be encoded using the terminal strongly-connected components of some population protocol. This seems unlikely, but the existence of multiple paths to **P** makes the possibility that clocked population protocols might have this power not as absurd as it might appear at first glance.

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 $\Omega?$ 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 [MS15] 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 §4.1, 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 §4.2, 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 [AAD$^+$06, SCWB08, MS15] 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.

We do not really expect a system to wait for an infinite amount of time before delivering a clock tick, any more than we expect a scheduler that must satisfy a fairness condition eventually to delay this beyond the lifetime of the universe. Instead, in both cases, we think of an unbounded delay as representing a time that is *long enough*, and clock ticks simply give a protocol the ability to outwait the potentially unbounded delays imposed by the scheduler. A practical question that we have not addressed is how to identify how long is long enough to wait to deliver a clock tick for a specific protocol running under a specific scheduling rule. The answer to this question will depend very much on the details of the protocol and scheduler, and in particular on the way in which the global fairness condition is enforced. In the case where global fairness is imposed through a known bound on the time that the adversary can delay reaching a particular configuration, it may be possible to compute a fixed, deterministic time at which a clock tick may be delivered safely. If the global fairness condition is less predictable, as in the case of randomized scheduling where it holds only with probability 1 in the limit, then it may only be possible to make weaker, probabilistic guarantees of correctness. The general question of predicting the running time of a given protocol under a given scheduling regime, even in special cases, is an interesting open problem that is sadly beyond the scope of this paper.

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.

Acknowledgments

I would like to thank the anonymous referees for their many helpful comments and suggestions of related work.

References

[AAD$^+$06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, pages 235–253, March 2006.

[AAE08] Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. *Distributed Computing*, 21(3):183–199, September 2008.

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, November 2007.

[Asp17] James Aspnes. Clocked population protocols. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, July 25-27, 2017*, pages 431–440. ACM, 2017.

[BBB13] Joffroy Beauquier, Peva Blanchard, and Janna Burman. Self-stabilizing leader election in population protocols over arbitrary communication graphs. In *International Conference On Principles Of Distributed Systems*, pages 38–52. Springer, 2013.

[BBBD16] Joffroy Beauquier, Peva Blanchard, Janna Burman, and Oksana Denysyuk. 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*, pages 20–35. Springer, 2016.

[BEJ19] Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of broadcast consensus protocols. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[CKS81] Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, January 1981.

[CMN$^+$10] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G Spirakis. 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*, pages 270–281. Springer, 2010.

[CMN$^+$11] Ioannis Chatzigiannakis, Othon Michail, Stavros Nikolaou, Andreas Pavlogiannis, and Paul G. Spirakis. Passively mobile communicating machines that use restricted space. *Theor. Comput. Sci.*, 412(46):6469–6483, 2011.

[FJ06] Michael Fischer and Hong Jiang. Self-stabilizing leader election in networks of finite-state anonymous agents. In *International Conference On Principles Of Distributed Systems*, pages 395–409. Springer, 2006.

[FMR68] Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Mathematical systems theory*, 2:265–283, 1968.

[GR09] Rachid Guerraoui and Eric Ruppert. Names trump malice: Tiny mobile agents can tolerate Byzantine failures. In *International Colloquium on Automata, Languages, and Programming*, pages 484–495. Springer, 2009.

[HL00] Joel David Hamkins and Andy Lewis. Infinite time turing machines. *The Journal of Symbolic Logic*, 65(02):567–604, 2000.

[Imm86] Neil Immerman. Relational queries computable in polynomial time. *Information and Control*, 68(1):86 – 104, 1986.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

[Jec02] Thomas Jech. *Set Theory: The Third Millenium Edition: revised and expanded.* Springer Monographs in Mathematics. Springer, 2002.

[MCS11] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. Mediated population protocols. *Theoretical Computer Science*, 412(22):2434 – 2450, 2011.

[Min61] Marvin L. Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.

[MS15] Othon Michail and Paul G Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81:1–10, 2015.

[SCWB08] David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *natural computing*, 7(4):615–633, 2008.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

[Tho54] James F. Thomson. Tasks and super-tasks. *Analysis*, 15(1):1–13, 1954.

[Var82] Moshe Y. Vardi. The complexity of relational query languages (extended abstract). In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing*, STOC '82, page 137146, New York, NY, USA, 1982. Association for Computing Machinery.

[vN23] Johann v. Neumann. Zur Einfűhrung der transfiniten Zahlen. *Acta Universitatis Szegediensis*, 1:199–208, 1923.