School of Computation, Information and Technology
Technische Universität München

# Efficient Analysis of Population Protocols and Chemical Reaction Networks

**Martin Helfrich**

Vollständiger Abdruck der von der School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

**Doktors der Naturwissenschaften (Dr. rer. nat.)**

genehmigten Dissertation.

Die Dissertation wurde am 21.06.2023 bei der Technischen Universität München eingereicht und durch die School of Computation, Information and Technology am 01.11.2023 angenommen.

# Abstract

*Population protocols* are a model of distributed computation by an arbitrary number of identical finite-state agents. The agents compute the answer to a question about their initial population by interacting in pairs until they reach a stable consensus. To automatically reason about the computation of population protocols, we introduce *stage graphs*: formal objects proving that the agents compute the correct answer for each of the infinitely many initial populations. Although every correct population protocol can be verified by a stage graph, deciding if a protocol is correct has high theoretical complexity. In practice however, we can efficiently verify many population protocols by automatically constructing a stage graph that resembles correctness proofs written by humans. The resulting stage graph helps to understand how the protocol works, how fast it is, and why it is correct or incorrect.

In the closely related *chemical reaction network* model, agents are molecules that randomly interact according to chemical reactions. The simulation-based analysis of this stochastic model quickly becomes infeasible if it requires a large number of simulations or if each simulation involves many interactions. To make the analysis more efficient, we introduce *segmental simulation*: an approximate simulation approach based on memoization. It speeds up the simulation process by reusing parts of previous simulations to generate new ones. We show how to combine segmental simulation with other approximate simulation approaches for even better performance. Together with a new fully automated hybrid simulation scheme, we can significantly speed up the generation of trajectories and correctly predict the transient behavior of complex stochastic systems.

# Zusammenfassung

*Populationsprotokolle* sind ein Modell der verteilten Berechnung durch eine beliebige Anzahl von identischen Agenten, die endlich viele Zuständen haben. Die Agenten berechnen die Antwort auf eine Frage über ihre initiale Population. Dafür interagieren sie paarweise bis ein stabiler Konsens herrscht. Um die Berechnung von Populationsprotokollen automatisch zu analysieren, führen wir *Phasengraphen* ein: formale Objekte, die beweisen, dass die Agenten für jede der unendlich vielen initialen Populationen die richtige Antwort berechnen. Obwohl jedes korrekte Populationsprotokoll durch einen Phasengraphen verifiziert werden kann, hat das Entscheiden, ob ein Protokoll korrekt ist, eine hohe theoretisch Komplexität. In der Praxis können wir jedoch viele Populationsprotokolle effizient verifizieren, indem wir automatisch einen Phasengraphen konstruieren, der Korrektheitsbeweisen von Menschen ähnelt. Der resultierende Phasengraph hilft zu verstehen, wie das Protokoll funktioniert, wie schnell es ist und warum es korrekt oder inkorrekt ist.

Im eng verwandten Modell der *chemischen Reaktionsnetzwerke* interagieren Moleküle zufällig gemäß chemischer Reaktionen. Die simulationsbasierte Analyse dieses stochastischen Modells wird schnell unpraktikabel, wenn eine große Anzahl von Simulationen erforderlich ist oder jede Simulation viele Interaktionen erfordert. Um die Analyse effizienter zu gestalten, führen wir die *Segmentsimulation* ein: eine approximative Simulationstechnik basierend auf Memoisation. Sie beschleunigt den Simulationsprozess, indem Teile früherer Simulationen wiederverwendet werden, um neue zu generieren. Wir zeigen, wie man die Segmentsimulation mit anderen approximativen Simulationsansätzen kombinieren kann, um eine noch bessere Leistung zu erzielen. Zusammen mit einem neuen, vollautomatischen Hybrid-Simulationsverfahren können wir die Generierung von Simulationen signifikant beschleunigen und das transiente Verhalten komplexer, stochastischer Systeme korrekt vorhersagen.

# Acknowledgments

First and foremost, I would like to express my deepest appreciation to my advisor, Javier Esparza, for his unwavering support and invaluable guidance. Javier's open-door policy and constant availability truly exemplified his dedication to my success. As a researcher, teacher, and team leader, he served as an inspiring role model, motivating me to strive for excellence in every aspect of my career. I am wholeheartedly convinced that I couldn't have found a more exceptional mentor.

I extend my sincere appreciation to my second supervisor, Jan Křetínský, for his significant contributions to my research in the fields of systems biology and chemistry. Jan's expertise and guidance played a pivotal role in steering me in the right direction throughout my research journey. I am particularly grateful for his remarkable ability to swiftly enhance and condense complex texts, greatly improving the clarity and effectiveness of my written work.

Furthermore, I would like to express my heartfelt appreciation to my co-authors at the Chair I7: Michael Blondin, Philipp Czerner, Roland Guttenberg, Clara Meyer, and Stefan Jaax. Their fruitful discussions and collaborations have been invaluable in shaping my research. I would also like to thank my co-author Milan Češka from Brno, whose steady support helped me overcome setbacks along the way. I am immensely grateful for all my past and present colleagues at I7 for the countless pleasant conversations and mutual support. A special and heartfelt shout-out goes to Maximilian Weininger for inspiring my creative endeavors.

I am especially grateful for my exceptional roommate Philipp. Our engaging discussions, both work-related and others, have been a constant source of inspiration and enjoyment. During our time as teaching assistants for the course THEO, we conceived innovative ideas that help students to learn and stay motivated. This accomplishment fills me with immense pride, and it would not have been possible without Philipp's indispensable technical know-how and dedication.

Finally, I would like to express my deep appreciation to my friends and family who patiently endured my explanations. Their willingness to listen, even when understanding only parts, has been immensely helpful in breaking down complex ideas. I want to thank my friends Vinzenz and Florian for taking the time to read my thesis and providing valuable feedback. My deepest appreciation goes to my partner Veronika for her unconditional support and encouragement in every aspect of my life. Similarly, I appreciate my mother for her influential support that shaped me into the scientist I am today. And lastly, I am grateful to my grandfather for being my biggest fan.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | |
|---|---|
| CRN | Chemical Reaction Network (see Chapter 6). |
| CTMC | Continuous-Time Markov Chain. |
| DTMC | Discrete-Time Markov Chain. |
| HYB | Hybrid Simulation (see Section 7.4). |
| HYBSEG | Hybrid Segmental Simulation. |
| LTL | Linear Temporal Logic (see Section 3.1). |
| ODE | Ordinary Differential Equation. |
| PP | Predator Prey (A CRN defined in Table 7.2). |
| RP | Repressilator (A CRN defined in Table 7.2). |
| SEG | Segmental simulation (see Chapter 7). |
| SSA | Stochastic Simulation Algorithm (see Section 6.3). |
| TS | Toggle Switch (A CRN defined in Table 7.2). |
| VI | Viral Infection (A CRN defined in Table 7.2). |

# 1 Introduction

*"The totality is not, as it were, a mere heap, but the whole is something besides the parts."*[1]

Aristotle, Metaphysics, 4th cent. BC

Observing the behavior of a single ant reveals little about the intricate dynamics of an entire ant colony, just as studying the flight patterns of individual birds fails to capture the mesmerizing coordination of a flock. These examples illustrate Aristotle's profound insight that the whole is more than just a collection of parts. This concept is especially relevant when trying to comprehend the dynamics of systems consisting of multiple interacting entities, such as *population protocols* and *chemical reaction networks*. Despite the fact that every entity follows simple rules, the interactions between the large number of entities give rise to global behavior that is challenging to analyze and understand.

In 2004, Angluin *et al.* considered the following scenario: We are tasked to detect whether there are at least five unhealthy birds in a flock. Each bird is equipped with a small sensor that can detect if the bird's temperature is elevated. These sensors have a limited amount of memory and can only communicate when two birds happen to be in close proximity. Despite these limitations, there is a simple protocol that first collects the information about the total number of unhealthy birds in a single sensor and then propagates whether there were at least five of them. Notably, this protocol works regardless of the number of healthy and unhealthy birds in the flock. Furthermore, there even is a protocol to determine if at least 5% of the birds are sick, although a single sensor cannot even count the total number of birds. Inspired by this computation in passively mobile sensor networks, Angluin *et al.* introduced a model of distributed computation known as *population protocols* [AAD+04; AAD+06]. In this model, identical finite-state agents interact in pairs to answer a question about the initial state of the system. The decision is made through stable consensus, where all agents eventually agree on the answer and then never change their minds. Similar to the protocol for counting unhealthy birds, population protocols are typically designed to answer a specific question. This is a challenging and error-prone task since they cannot be programmed directly but are defined through a list of rendezvous transitions. Naturally, we would like to analyze the behavior of a population protocol and even verify that it works as intended. However, similar to the study of an ant colony through a single ant, we only know how two agents interact but need to argue about the behavior of an arbi-

[1] This quote from Aristotle's Metaphysics is a translation by W. D. Ross [Ari85, 1045a7-1045a20]. It is often misquoted as: *"The whole is more than the sum of its parts."*

trary number of agents. In this work, we introduce a technique for efficiently verifying population protocols based on stage graphs, formal objects that prove the correctness of a protocol and provide valuable insights into its computation process.

In order to obtain a comprehensive understanding of distributed computation, researchers have investigated multiple theoretical aspects of population protocols. These include the class of questions they can answer (expressive power), the number of interactions needed to compute the answer (speed), and the minimal number of states the agents require (size). A central result is that population protocols can precisely answer questions that can be defined in Presburger arithmetic [AAE+07]. The proof of this result on the expressive power even gives a synthesis procedure that constructs a protocol for every possible Presburger formula. However, while the resulting protocol is fast, its size grows exponentially with the length of the formula. As this thesis focuses on the efficient analysis of population protocols, we provide only a high-level overview of our related work on the synthesis of population protocols that are both fast and have a small size.

Population protocols are closely related to *chemical reaction networks*, a model describing molecules in a well-mixed solution that interact through chemical reactions. Chemical reaction networks can be understood as a continuous-time variant of population protocols where the number of agents can vary during the evolution of the system. They are used to model stochastic systems with applications in biochemistry [CBH+09] but also in epidemiology [LMV22] and molecular programming [SSW10]. Analyzing the properties and behavior of these modeled systems poses significant challenges: Like for population protocols, we have only access to a list of possible chemical reactions. Moreover, because the number of molecules can grow, the state space can be infinite. One possible approach to address this problem is to estimate system properties by sampling trajectories. However, this simulation-based analysis often requires a large number of simulations to gather sufficient statistics about the system's behavior. The process can be particularly slow when each simulation involves sampling a high number of interactions. To overcome these limitations, we propose a new approximate simulation technique called segmental simulation which is based on the concept of memoization[2]. Segmental simulation accelerates the generation of new simulations by reusing parts of previous simulations, thereby enabling efficient analysis of complex chemical reaction networks.

## 1.1 Literature Overview

We provide a chronological overview of the literature that led to the research in this thesis and focus on related models and research directions. A more detailed discussion of work related to our methods and contributions is given in Section 4.4 for the verification of population protocols and in Section 7.7 for the simulation of chemical reaction networks.

[2]To clarify, *memoization* (without 'r') is an optimization technique that stores results to avoid redundant calculations, while *memorization* (with 'r') refers to the process of committing something to memory.

The research on the analysis of chemical reaction networks began in 1931 with Kolmogorov's description of the time evolution of a stochastic jump process through differential equations [Kol31]. This description, known today as the chemical master equation [Gil92], became the foundation for the first stochastic simulation algorithm for chemical reaction networks, developed by Doob in 1945 [Doo45]. In 1950, the approach was implemented on a computer [Ken50], enabling the study of outbreaks in epidemics three years later [Bar53]. In 1977, Gillespie popularized the stochastic simulation algorithm (SSA), which serves as the basis for most simulation-based analysis approaches [Gil77]. An alternative approach is the numerical analysis of chemical reaction networks, which directly computes transient distributions. The commonly used technique with the name *uniformization* transforms the continuous-time model into a discrete-time model with a uniform time step [Gra77; GM84]. In order to handle the (potentially infinite) state space resulting from this transformation, various state-reduction techniques have been explored [MK06; ZWC09; AAČ+21]. Similarly, abstraction-based methods construct a smaller model while preserving the dynamical properties of the original chemical reaction network [ČK19].

The research on population protocols was inspired by work in the early 2000s on sensor networks [IGE00] and trust propagation [DF01]. In 2004, Angluin *et al.* introduced the population protocol model in their seminal paper [AAD+04; AAD+06]. The connection to chemical reaction networks was only discovered a few years later [AAE07; AR09]. Population protocols are closely related to several other theoretical models studied in computer science, including Petri nets [DA94], multiset rewriting systems [BT05], and vector addition systems [KM69]. Consequently, results from these related models can be applied to research on population protocols. For example, Esparza *et al.* establish a connection between population protocols and Petri nets, demonstrating the decidability of reachability and correctness problems for population protocols [EGL+17]. A fundamental result characterizing the expressive power of population protocols shows that they can precisely compute the class of formulas in Presburger arithmetic, the first-order theory of addition [AAE+07]. Other research investigates the speed of computation [AAE08; AGV15], such as studies on fast leader election [DS18], and space complexity [BEJ18a], including the design of highly succinct threshold protocols where agents have few states but can count to large numbers [CE21]. Numerous modifications of the population protocol model have been explored, such as protocols with faulty interactions [DFI+19] and protocols with leaders [AAE08; DFI+19]. Additional variations alter the capabilities of agents by incorporating a global clock [Asp17; MS15a; GS20], introducing identifiers [GR07; MCS11], or allowing the number of states to depend on the number of agents [AAE+17; GS20]. Similarly, other publications modify the communication between agents, for example, by adding broadcast transitions [BEJ19], by limiting interactions to observation [EGM+18], or by introducing different network topologies [BBB13; CFQ+12].

## 1.2 Summary of Contributions

We give a brief and high-level overview of the main contributions. Additional details can be found in Chapters 3 to 5.

**Verification of Population Protocols.** In [BEH+20], we present a sound and complete method for the verification of population protocols that uses Presburger stage graphs. Presburger stage graphs formally describe the computation of population protocols as a series of stages that correspond to irreversible changes in the system state. As such, they can act as a witness for liveness properties like correctness and can be checked independently. Further, we show that if a population protocol is correct, then there is a Presburger stage graph that proves this. Although this yields an algorithm for the verification of arbitrary population protocols, it is not very efficient due to the high theoretical complexity of the verification problem. Thus, we introduce an incomplete but efficient procedure that constructs Presburger stage graphs that can verify many population protocols from the literature. In [EHJ+20], we extend the tool Peregrine with this new automatic verification procedure together with a visualization of the generated stage graphs. This helps users to understand how a protocol works, how fast it is, and why it is correct or incorrect.

**Synthesis of Fast and Succinct Population Protocols.** In [BEG+20], we present the first synthesis procedure that results in succinct population protocols, i.e., protocols with agents that have few states. Specifically, for a given Presburger formula $\varphi$ we produce a population protocol with agents that have $\text{poly}(|\varphi|)$ states where $|\varphi|$ is the length of the formula with binary coefficients. In [CGH+22], we improve the result by producing succinct population protocols that are also fast. Specifically, the expected number of interactions required to reach a stable consensus in the resulting protocols is $\mathcal{O}(n^2)$ where $n$ is the number of agents.

**Approximate Simulation of Chemical Reaction Networks.** In [HČK+22], we introduce segmental simulation, a novel approximate simulation method for chemical reaction networks based on memoization. It speeds up the generation of new simulations by reusing small parts of previous simulations that are called segments. To ensure that segmental simulations have the same behavior as the original systems, we leverage a population-level abstraction that divides the state space into regions with similar local behavior. This allows us to only reuse segments that start in the same region. In addition to a discussion of the approximation error, we demonstrate the accuracy and speedup of the approach using a series of benchmarks. For even more efficiency, we show how to combine segmental simulation with other approximate simulation techniques. To this end, we present a new hybrid simulation approach that automatically classifies the speed of reactions. This allows us to approximate the effect of a large number of fast reactions while preserving the stochastic fluctuations caused by slow reactions. We implemented our approaches in the easy-to-use tool SAQuaiA which helps to simulate and analyze chemical reaction networks and facilitates the comparison of different approximate simulation techniques.

## 1.3 Publication Summary

We list the publications by the thesis author that we discuss in this publication-based thesis. All papers are included in the appendix, preceded by a page that presents the full citation, a short summary, and a description of the thesis author's contributions.

Part I of the appendix contains three publications in which the author of this thesis is the **first author**. In these publications, the thesis author contributed more than 50% of the substantive findings:

A Michael Blondin, Javier Esparza, Martin Helfrich, Antonín Kučera and Philipp J. Meyer. “Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling”.
CAV, 2020. [BEH+20]

B Javier Esparza, Martin Helfrich, Stefan Jaax and Philipp J. Meyer. “Peregrine 2.0: Explaining Correctness of Population Protocols through Stage Graphs”.
ATVA, 2020. [EHJ+20]

C Martin Helfrich, Milan Češka, Jan Křetínský and Štefan Martiček. “Abstraction-Based Segmental Simulation of Chemical Reaction Networks”.
CMSB, 2022. [HČK+22]

Note that only publications A and C are considered **core publications**, as they are full publications, while B is a tool paper.

Part II of the appendix contains the following two publications in which the author of this thesis is **not the first author**:

D Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich and Stefan Jaax. “Succinct Population Protocols for Presburger Arithmetic”.
STACS, 2020. [BEG+20]

E Philipp Czerner, Roland Guttenberg, Martin Helfrich and Javier Esparza. “Fast and Succinct Population Protocols for Presburger Arithmetic”.
SAND, 2022. [CGH+22]

All five publications included in the dissertation are written in English and have been published in peer-reviewed proceedings of internationally recognized conferences. Publications A and B are covered by Chapter 4, publication C is covered by Chapter 7, and a high-level overview of publications D and E is presented in Chapter 5.

### Other Publications

In addition to the included publications, the author has co-authored the following papers while working on this thesis. These papers have been published in peer-reviewed conference proceedings and, while they are not part of the thesis, they are mentioned here for the sake of completeness.

- Loris D'Antoni, Martin Helfrich, Jan Kretinsky, Emanuel Ramneantu and Maximilian Weininger. "Automata Tutor v3".
  CAV, 2020. [DHK+20]
- Philipp Czerner, Roland Guttenberg, Martin Helfrich and Javier Esparza. "Decision Power of Weak Asynchronous Models of Distributed Computing".
  PODC, 2021. [CGH+21]

## 1.4 Outline of Thesis

Chapter 2 provides an overview of preliminary concepts and notations related to numbers, vectors, multisets, and Presburger arithmetic.

Chapter 3 focuses on the model of population protocols, presenting example protocols to illustrate their workings. The verification of population protocols is covered in Chapter 4, where the results from [BEH+20; EHJ+20] are presented. Furthermore, Chapter 5 provides a concise and high-level overview of our complementary research concerning the synthesis of fast and succinct population protocols in [BEG+20; CGH+22].

In Chapter 6, we introduce the chemical reaction network model. Building on this, Chapter 7 delves into the results obtained from [HČK+22] for the approximate simulation of chemical reaction networks.

# 2 Preliminaries

$\mathbb{B} \overset{\text{def}}{=} \{\mathit{true}, \mathit{false}\}$ are the Boolean values, $\mathbb{N} \overset{\text{def}}{=} \{0, 1, \dots\}$ are the natural numbers, $\mathbb{Z} \overset{\text{def}}{=} \{\dots, -1, 0, 1, \dots\}$ are the integers, and $\mathbb{R}$ is the set of real numbers.

**Vectors.** Let $E$ be a finite set. A *vector* $V$ over $E$ is a mapping $V : E \mapsto \mathbb{R}$. The set of all vectors over $E$ is $\mathbb{R}^E$. Addition, subtraction, and scalar multiplication of vectors are defined componentwise, i.e., $(M + N)(e) \overset{\text{def}}{=} M(e) + N(e)$, $(M - N)(e) \overset{\text{def}}{=} M(e) - N(e)$, and $(c \cdot M)(e) \overset{\text{def}}{=} c \cdot M(e)$ for $M, N \in \mathbb{Z}^E$ and $c \in \mathbb{Z}$. The dot product of two vectors $M, N \in \mathbb{Z}^E$ is defined as $M \cdot N \overset{\text{def}}{=} \sum_{e \in E} M(e) + N(e)$. Comparison of vectors is defined as a partial order with $M \geq N \overset{\text{def}}{\iff} M(e) \geq N(e)$ for every $e \in E$.

**Multisets.** Let $E$ be a finite set. A *multiset* $M$ over $E$ is a vector $M : E \mapsto \mathbb{N}$ and the set of all multisets over $E$ is $\mathbb{N}^E$. $M(e)$ is the number of occurrences of $e$ in multiset $M$. The *support* and *size* of a multiset $M$ are $\mathrm{supp}(M) \overset{\text{def}}{=} \{e \in E \mid M(e) > 0\}$ and $|M| \overset{\text{def}}{=} \sum_{e \in E} M(e)$, respectively. $E^{\langle k \rangle}$ is the set of all multisets over $E$ of size $k$. In contrast to vector subtraction $(-)$, we define multiset difference $(\ominus)$ as $(M \ominus N)(e) \overset{\text{def}}{=} \max(M(e) - N(e), 0)$ for $M, N \in \mathbb{N}^E$. We often use a set-like notation to describe multisets, e.g., $M = \wr a, a, b \wr$, or equivalently $M = \wr 2 \cdot a, b \wr$, is the multiset with $M(a) = 2$, $M(b) = 1$ and $M(x) = 0$ for all $x \notin \{a, b\}$.

**Presburger arithmetic.** Let $X$ be a set of variables. The set of *Presburger formulas* over $X$ is the result of closing atomic formulas, as defined in the following sentence, under Boolean operations and first-order existential quantification. *Atomic formulas* are of the form $\sum_{i=1}^{k} a_i x_i \sim b$, where $a_i$ and $b$ are integers, $x_i$ are variables, and $\sim$ is either $<$ or $\equiv_m$, the latter denoting the congruence modulo $m$ for any $m \geq 2$. Formulas over $X$ are interpreted on $\mathbb{N}^X$, i.e., variables take values in the natural numbers. Let $\varphi$ be a formula of Presburger arithmetic over $X$. If the multiset $E \in \mathbb{N}^X$ satisfies the Presburger formula $\varphi$ over $X$, we write $\varphi(M) = \mathit{true}$ or just $\varphi(M)$, otherwise $\varphi(M) = \mathit{false}$. For example, the Presburger formula $\varphi = (2x + 3y < 10) \land (x + y \equiv_2 0)$ is satisfied for multiset $M = \wr 3 \cdot x, y \wr$ because $2 \cdot 3 + 3 \cdot 1 = 9 < 10$ and $3 + 1 \equiv_2 0$, i.e., we write $\varphi(M)$. Let $[\![\varphi]\!]$ be the set of all multisets satisfying $\varphi$. A set of multisets $\mathscr{C} \subseteq \mathbb{N}^X$ is a *Presburger set* if $\mathscr{C} = [\![\varphi]\!]$ for some formula $\varphi$. We say a set of multisets $\mathscr{C} \subseteq \mathbb{N}^X$ satisfies a Presburger formula $\varphi$ over $X$ if $\mathscr{C} \subseteq [\![\varphi]\!]$. The *size* $|\varphi|$ of a Presburger formula $\varphi$ is the length of its string representation with coefficients written in binary. For a survey on Presburger arithmetic, see [Haa18].

# 3 Population Protocols

In this section, we will introduce the population protocol model. We begin with a high-level explanation to give a first intuition and then give a formal introduction. For a survey on population protocols, we refer to [AR09].

In 2004 Angluin *et al.* introduced population protocols as a model of distributed computation by a collection of agents [AAD+04; AAD+06]. Each agent is in one of finitely many states but is otherwise indistinguishable from other agents. As such, the global state of the population protocol, called its configuration, is fully determined by the number of agents in each state. The agents compute the answer to a question about their initial population by interacting in pairs until they reach a stable consensus. In every discrete time step, a fair scheduler picks the next pair of agents to interact, resulting in an infinite sequence of configurations. Depending on its current state, each agent has an opinion on whether the answer is true or false. If all agents have the same opinion, there is a consensus. If, additionally, all possible future configurations have the same consensus, there is a stable consensus. We say that a protocol computes a question if any fair execution reaches a stable consensus with the right opinion. Our running example will be the majority protocol:

**Example 1** (Majority Protocol)**.** *The majority protocol has four states $Y, N, y$, and $n$. Agents start in one of the two initial states $Y$ and $N$ with the idea that agents in $Y$ vote for "yes" and agents in $N$ vote for "no". If the system starts with three agents in $Y$ and two agents in $N$, we say that its initial configuration is the multiset $C_0 = \wr Y, Y, Y, N, N \wr$. The protocol has four transitions that change the state of agents:*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad t_4 : y, n \mapsto y, y$$

*Intuitively, transition $t_1$ tells us that if an agent in $Y$ and an agent in $N$ interact, they can change their state: one agent to $y$ and the other agent to $n$. In $C_0$ the transition $t_1$ leads to $\wr Y, Y, y, N, n \wr$. See Figure 3.1 for a visualization of the states and transitions as a Petri net.*

*The majority protocol computes the answer to the Presburger formula $Y \geq N$, i.e., it computes whether there are at least as many votes for "yes" as for "no" in the initial configuration. The current opinion of an agent is "yes" if its state is $Y$ or $y$, and "no" otherwise. Because in the initial configuration $C_0$ there are indeed at least as many agents in $Y$ as in $N$, every agent will eventually have the opinion "yes" forever. Otherwise, every agent would eventually have the opinion "no" forever.*

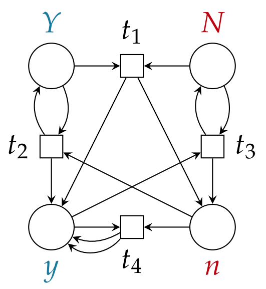


**Figure 3.1:** Petri net visualization of majority population protocol defined in Example 1. The places (circles) of the Petri net represent states, and the transitions of the Petri net (squares) correspond to the transitions in the protocol.

## 3.1 Formal Definition with Examples

A population protocol is a tuple $\mathcal{P} = (Q, T, I, O)$, where

- $Q$ is a finite set of *states*,
- $T \subseteq Q^{\langle 2 \rangle} \times Q^{\langle 2 \rangle}$ is a set of *transitions* containing the set $\{(x, x) \mid x \in Q^{\langle 2 \rangle}\}$ of *silent* transitions,[1]
- $I \subseteq Q$ is the set of *initial states*, and
- $O : Q \mapsto \mathbb{B}$ is the *output function*.

**Example 2** (Formal Definition of Majority Protocol)**.** *The majority protocol is the tuple* $(Q, T, I, O)$ *with*

$$Q \overset{def}{=} \{Y, N, y, n\} \qquad I \overset{def}{=} \{Y, N\} \qquad O(s) = 1 \overset{def}{\iff} s \in \{Y, y\}$$

*and T contains the four non-silent transitions:*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad t_4 : y, n \mapsto y, y$$

**Configurations and Transitions.** A *configuration* is a multiset of states $C \in \mathbb{N}^Q$ with $|C| \geq 2$, where $C(q)$ is the number of agents in state $q \in Q$. Because agents are indistinguishable, the configuration fully determines the global state of the system. If $C(q) = 0$, then state $q$ is *empty* in $C$. For a transition $t = (\wr q_1, q_2 \wr, \wr q_3, q_4 \wr)$ we usually write $t : q_1, q_2 \mapsto q_3, q_4$. The *preset*, *postset*, and *effect* are defined as ${}^\bullet t \overset{\text{def}}{=} \wr q_1, q_2 \wr$, $t^\bullet \overset{\text{def}}{=} \wr q_3, q_4 \wr$, and $\Delta(t) \overset{\text{def}}{=} t^\bullet - {}^\bullet t$, respectively. A transition $t$ is *enabled* in $C$ if $C \geq {}^\bullet t$ and *disabled* otherwise. If $t$ is enabled in $C$, it can *occur* leading to configuration $C' = C + \Delta(t)$ which

[1]When giving population protocols we usually omit silent transitions but implicitly assume they are present.

we write as $C \xrightarrow{t} C'$. Intuitively, when $q_1, q_2 \mapsto q_3, q_4$ occurs, then two agents in states $q_1$ and $q_2$ interact and change their state to $q_3$ and $q_4$. Note that transitions cannot create or destroy agents.

**Reachability, Executions, and Fairness.** Let $C$ and $C'$ be configurations. We say $C$ can *reach $C'$ in one step*, written as $C \rightarrow C'$, if $C \xrightarrow{t} C'$ for some transition $t$. Further, $C$ can *reach $C'$ via a transition sequence $w = t_0, t_1, \ldots, t_n$*, written as $C \xrightarrow{w} C'$, if there are configurations $C_0, C_1, C_2, \ldots, C_n$ such that $C = C_0$ and $C_n = C'$ and $C_0 \xrightarrow{t_0} C_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{n-1}} C_n$. Finally, we say $C$ can *reach* $C'$, written as $C \xrightarrow{*} C'$, if there is a transition sequence $w$ such that $C \xrightarrow{w} C'$. For a set of configurations $\mathscr{C}$, let $post^*(\mathscr{C}) \overset{\text{def}}{=} \{C' \in \mathbb{N}^Q \mid C \in \mathscr{C}, C \xrightarrow{*} C'\}$ denote the set of *reachable* configurations. An *execution* starting in configuration $C$ is an infinite sequence of configurations $\pi = C_0 C_1 C_2 \ldots$ such that $C = C_0$ and $C_i \rightarrow C_{i+1}$ for every $i$. An execution is *fair* if every configuration that is reachable infinitely often is visited infinitely often. More details on fairness are given in Section 3.3.

**Example 3** (Reachability Graph for Majority Protocol). *For some fixed configuration, the number of reachable configurations is always finite because the number of agents stays constant. Thus, it is possible to visualize the set of reachable configurations as a finite reachability graph. The reachability graph for the configuration $C = \Lbag Y, Y, Y, N, N \Rbag$ in the majority protocol is shown in Figure 3.2. Note that every fair execution starting at $C$ will eventually visit $C' = \Lbag Y, y, y, y, y \Rbag$ as this configuration is always reachable. As only silent transitions are enabled in $C'$, the system will remain in $C'$ forever. Without fairness, an execution could, for example, change between $\Lbag Y, Y, y, N, n \Rbag$ and $\Lbag Y, Y, y, y, N \Rbag$ forever.*

$$\begin{tikzcd}
 & & \Lbag Y,Y,Y,N,N \Rbag \arrow[d, "t_1"] & \\
 & \Lbag Y,Y,y,y,N \Rbag \arrow[r, bend left, "t_3"] \arrow[d, "t_1"] & \Lbag Y,Y,y,N,n \Rbag \arrow[l, bend left, "t_2 \,/\, t_4"] \arrow[r, bend left, "t_3"] \arrow[d, "t_1"] & \Lbag Y,Y,N,n,n \Rbag \arrow[l, bend left, "t_2"] \arrow[d, "t_1"] \\
\Lbag Y,y,y,y,y \Rbag & \Lbag Y,y,y,y,n \Rbag \arrow[l, "t_2 \,/\, t_4"'] & \Lbag Y,y,y,n,n \Rbag \arrow[l, "t_2 \,/\, t_4"'] & \Lbag Y,y,n,n,n \Rbag \arrow[l, "t_2 \,/\, t_4"']
\end{tikzcd}$$

**Figure 3.2:** Finite reachability graph of configuration $\Lbag Y, Y, Y, N, N \Rbag$ in the majority protocol. All loops corresponding to silent transitions were omitted. Please note that there are infinitely many configurations because the number of initial agents is unbounded. For each of these configurations there is such a bounded reachability graph.

**LTL.** We now lift the notion of Presburger formulas as defined in Section 2 to linear temporal logic (LTL). Formulas of LTL over Presburger atomic propositions are given by the grammar

$$\psi \coloneqq \varphi \mid \neg\psi \mid \psi \vee \psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi \,\mathbf{U}\, \psi$$

where $\varphi$ is a Presburger formula. We use the standard semantics:

$$\begin{array}{lcl}
(\neg\psi)(C) & \Leftrightarrow & \neg\psi(C) \\
(\psi_1 \wedge \psi_2)(C) & \Leftrightarrow & \psi_1(C) \wedge \psi_2(C) \\
(\psi_1 \vee \psi_2)(C) & \Leftrightarrow & \psi_1(C) \vee \psi_2(C) \\
(\mathbf{X}\psi)(C) & \Leftrightarrow & \forall C' : C \to C' \Rightarrow \psi(C') \\
(\psi_1 \,\mathbf{U}\, \psi_2)(C) & \Leftrightarrow & \forall \text{ fair } C_0C_1\ldots \text{ with } C_0 = C : \exists i > 0 : \\
 & & \qquad \psi_2(C_i) \wedge \bigwedge_{j<i} \psi_1(C_j)
\end{array}$$

Intuitively, **X** stands for "next" as the following property holds in every successor configuration, and **U** stands for "until" because in fair executions the first property holds up to a point where the second property holds. Furthermore, we define the abbreviations $\mathbf{F}\psi \overset{\text{def}}{=} \mathit{true} \,\mathbf{U}\, \psi$ and $\mathbf{G}\psi \overset{\text{def}}{=} \neg\mathbf{F}\neg\psi$. Intuitively, **F** stands for "finally" and forces that the following property holds at some point in the future, whereas **G** stands for "globally" and holds if the property holds now and in every future configuration. Like for Presburger formulas, $[\![\psi]\!]$ is the set of configurations that satisfy $\psi$. A set of configurations $\mathscr{C} \subseteq \mathbb{N}^Q$ satisfies $\psi$ if $\mathscr{C} \subseteq [\![\psi]\!]$. A population protocol $\mathcal{P}$ satisfies $\psi$ if $\mathbb{N}^Q = [\![\psi]\!]$, i.e., if every configuration satisfies the property.

**Computation.** An *initial configuration* is a configuration that satisfies the Presburger formula $\varphi_{\text{init}} \overset{\text{def}}{=} \bigwedge_{q \in Q \setminus I} q{=}0$, i.e., a configuration where non-initial states are empty.[2] A configuration $C$ has *consensus* $b \in \mathbb{B}$ if it satisfies the formula $\varphi^b_{\text{cons}} \overset{\text{def}}{=} \bigwedge_{q \in Q}(q > 0 \implies O(q) = b)$, i.e., if every agent has output $b$. A configuration is a *stable consensus* with output $b$ if it satisfies $\mathbf{G}\varphi^b_{\text{cons}}$, i.e., if any reachable configuration has consensus $b$. A population protocol *computes* a Presburger formula $\varphi : \mathbb{N}^I \mapsto \mathbb{B}$ if any fair execution starting an initial configuration $C_0$ eventually stabilizes to a consensus with output $\varphi(C_0)$, i.e., if the protocol satisfies the LTL formula

$$(\varphi_{\text{init}} \wedge \varphi \implies \mathbf{FG}\varphi^{\mathit{true}}_{\text{cons}}) \wedge (\varphi_{\text{init}} \wedge \neg\varphi \implies \mathbf{FG}\varphi^{\mathit{false}}_{\text{cons}})$$

It is known that population protocols compute precisely the formulas expressible in Presburger arithmetic [AAE+07]: Every formula that is computed by a population protocol is Presburger, and for every Presburger formula there is a population protocol that computes it.

[2]Note that population protocols are traditionally defined with an additional input mapping that maps some input alphabet to initial states. Our definition is equivalent, as one can always introduce states for each input letter and perform the mapping via auxiliary transitions.

**Example 4** (Computation for Majority Protocol)**.** *For the majority protocol, the formulas for initial configurations and configurations with consensus are:*

$$\begin{aligned}\varphi_{\text{init}} &= y = 0 \land n = 0\\ \varphi_{\text{cons}}^{\text{true}} &= N = 0 \land n = 0\\ \varphi_{\text{cons}}^{\text{false}} &= Y = 0 \land y = 0\end{aligned}$$

*The majority protocol computes the formula $Y \geq N$, i.e., it satisfies:*

$$(\varphi_{\text{init}} \land Y \geq N \implies \mathbf{FG}\varphi_{\text{cons}}^{\text{true}}) \land (\varphi_{\text{init}} \land Y < N \implies \mathbf{FG}\varphi_{\text{cons}}^{\text{false}})$$

*In other words: The majority protocol starts with all agents in $Y$ or $N$. If there was an initial majority for $Y$ (or a tie), then all agents eventually stabilize to output* true*. Otherwise, they eventually stabilize to output* false*.*

## 3.2 More Examples

**Example 5** (Incorrect Majority Protocol)**.** *If we remove the transition $t_4$ of the majority protocol of Example 1, the resulting population protocol no longer computes $Y \geq N$.*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad \cancel{t_4 : y, n \mapsto y, y}$$

*In fact, if there is a tie, the protocol does not even reach a consensus. The smallest example is the execution $\wr Y, N \wr \xrightarrow{t_1} \wr y, n \wr \rightarrow \wr y, n \wr \rightarrow \cdots$. Note that this execution is fair as (i) $\wr y, n \wr$ is the only configuration that is visited infinitely, and (ii) it enables only silent transitions. Because of this, the protocol does not compute any formula.*

**Example 6** (Approximate Majority)**.** *Consider the population protocol $(Q, T, I, O)$ with*

$$Q \overset{\text{def}}{=} \{Y, N\} \qquad I \overset{\text{def}}{=} Q \qquad O(s) \overset{\text{def}}{=} (s = Y)$$

*and two non-silent transitions:*

$$t_Y : Y, N \mapsto Y, Y \qquad t_N : Y, N \mapsto N, N$$

*This protocol also does not compute $Y \geq N$. Consider that any configuration without consensus can be stabilized to* either *output. Using $t_Y$ a single agent in $Y$ can change the state of every agent in $N$ to $Y$. Similarly, using $t_N$ a single agent in $N$ can change the state of every agent in $Y$ to $N$. Thus, the protocol does not compute any formula because the output is not fully determined by the initial configuration. However, the protocol always stabilizes, i.e., it satisfies $\mathbf{F}(\mathbf{G}\varphi_{\text{cons}}^{\text{true}} \lor \mathbf{G}\varphi_{\text{cons}}^{\text{false}})$. Further, it can be shown that if there is a majority, the corresponding output is more likely under stochastic scheduling (see Section 3.3).*

**Example 7** (Flock-of-Birds Protocol)**.** *Assume that we have a flock with an unknown number of birds. Each bird has a sensor that detects if it is "sick" ($q_1$) or "healthy" ($q_0$). Now we will* *give a family of population protocols that computes if the number of sick birds is at least c for some parameter $c \geq 1$. Note that the example in the introduction of this thesis is the protocol for $c = 5$ (see Chapter 1).*

*The flock-of-birds protocol for $c \geq 1$ is the protocol $(Q, T, I, O)$ with:*

$$Q \overset{def}{=} \{q_0, q_1, q_2, \dots, q_c\} \qquad I \overset{def}{=} \{q_0, q_1\} \qquad O(s) \overset{def}{=} (s = q_c)$$
$$T \overset{def}{=} \{q_x, q_y \mapsto q_{x+y}, q_0 \mid x + y < c\} \cup \{q_x, q_y \mapsto q_c, q_c \mid x + y \geq c\}$$

*The protocol computes the formula $q_1 \geq c$. Intuitively, an agent in state $q_x$ knows that x agents are sick. When two agents in state $q_x$ and $q_y$ interact, they together know that $x + y$ birds are sick. If $x + y \geq c$ they know that there are indeed at least c sick birds and both go to state $q_c$. Otherwise, one of the birds collects all the information by changing its state to $q_{x+y}$ while the other agent pretends that everyone is healthy.*

## 3.3 Other Fairness Notions and Stochastic Scheduling

Intuitively, the next interaction is chosen by a scheduler. The scheduler is free to choose any interaction, even in an adversarial manner, as long as the resulting sequence of interactions is considered fair. However, there are multiple different fairness notions, that we will explain in this section.

**Global Fairness.** Recall that we use a global fairness assumption in the definition of population protocols. An execution is *fair* if all configurations that can be reached infinitely often are reached infinitely often. There is an equivalent alternative definition of global fairness: An execution is *one-step fair* if all configurations that can be reached infinitely often in one step are reached infinitely often. While the one-step fairness definition may be more intuitively understandable, one typically uses the stronger global fairness to simplify proofs.

**Local Fairness.** In some publications on population protocols, global fairness is replaced by a local fairness notion [SLD+09; CDF+11]. An execution is *locally fair* if all transitions that are enabled infinitely often occur infinitely often. Note that local fairness and global fairness are incomparable, i.e., there are executions that are globally fair but not locally fair (see Figure 3.3) and vice versa (see Figure 3.4).

**Stochastic Scheduling.** Often, the fairness assumption is replaced by a stochastic scheduler that picks the next pair of interacting agents and the transition they perform uniformly at random (e.g., see [AAE08] or [BEK18]). This closely corresponds to the notion that agents are passively mobile sensors that move randomly in space and interact with other sensors when they are in close proximity (see Figure 3.5). It is easy to show that a stochastic scheduler is both (i) locally fair and (ii) globally fair: (i) If a transition is enabled infinitely often, it will almost surely occur infinitely often because

$$\begin{aligned} Q &\overset{\text{def}}{=} \{a, b, c\} \\ T &\overset{\text{def}}{=} \{t_{ab} : a, a \mapsto b, b \\ &\qquad t_{bc} : b, b \mapsto c, c \\ &\qquad t_{ca} : c, c \mapsto a, a \\ &\qquad t_{ac} : a, a \mapsto c, c\} \end{aligned}$$

$$\begin{tikzcd}
& \{\!\{a, a\}\!\} \arrow[dr, bend left, "t_{ab}"] \arrow[dl, dashed, "t_{ac}"] & \\
\{\!\{c, c\}\!\} \arrow[ur, bend left, "t_{ca}"] & & \{\!\{b, b\}\!\} \arrow[ll, bend left, "t_{bc}"]
\end{tikzcd}$$

**Figure 3.3:** Global fairness does not imply local fairness. An execution of the given population protocol is drawn as a solid orange cycle within a reachability graph. It is globally fair as it visits all reachable configurations infinitely often. However, it is not locally fair because the transition $t_{ac}$ is enabled infinitely often in $\{\!\{a, a\}\!\}$ but never occurs.

$$\begin{aligned} Q &\overset{\text{def}}{=} \{Y, N\} \\ T &\overset{\text{def}}{=} \{t_Y : Y, N \mapsto Y, Y \\ &\qquad t_N : Y, N \mapsto N, N\} \end{aligned}$$

$$\begin{tikzcd}
\{\!\{3 \cdot Y\}\!\} \arrow[r, bend left, dashed, "t_N"] & \{\!\{2 \cdot Y, N\}\!\} \arrow[l, bend left, dashed, "t_Y"] \arrow[r, bend left, "t_N"] & \{\!\{Y, 2 \cdot N\}\!\} \arrow[l, bend left, "t_Y"] \arrow[r, bend left, dashed, "t_N"] & \{\!\{3 \cdot N\}\!\} \arrow[l, bend left, dashed, "t_Y"]
\end{tikzcd}$$

**Figure 3.4:** Local fairness does not imply global fairness. An execution of the approximate majority protocol of Example 6 is drawn as a solid orange cycle within a reachability graph. It is locally fair as all transitions occur infinitely often. However, it is not globally fair because $\{\!\{3 \cdot Y\}\!\}$ is reachable infinitely often but never reached.

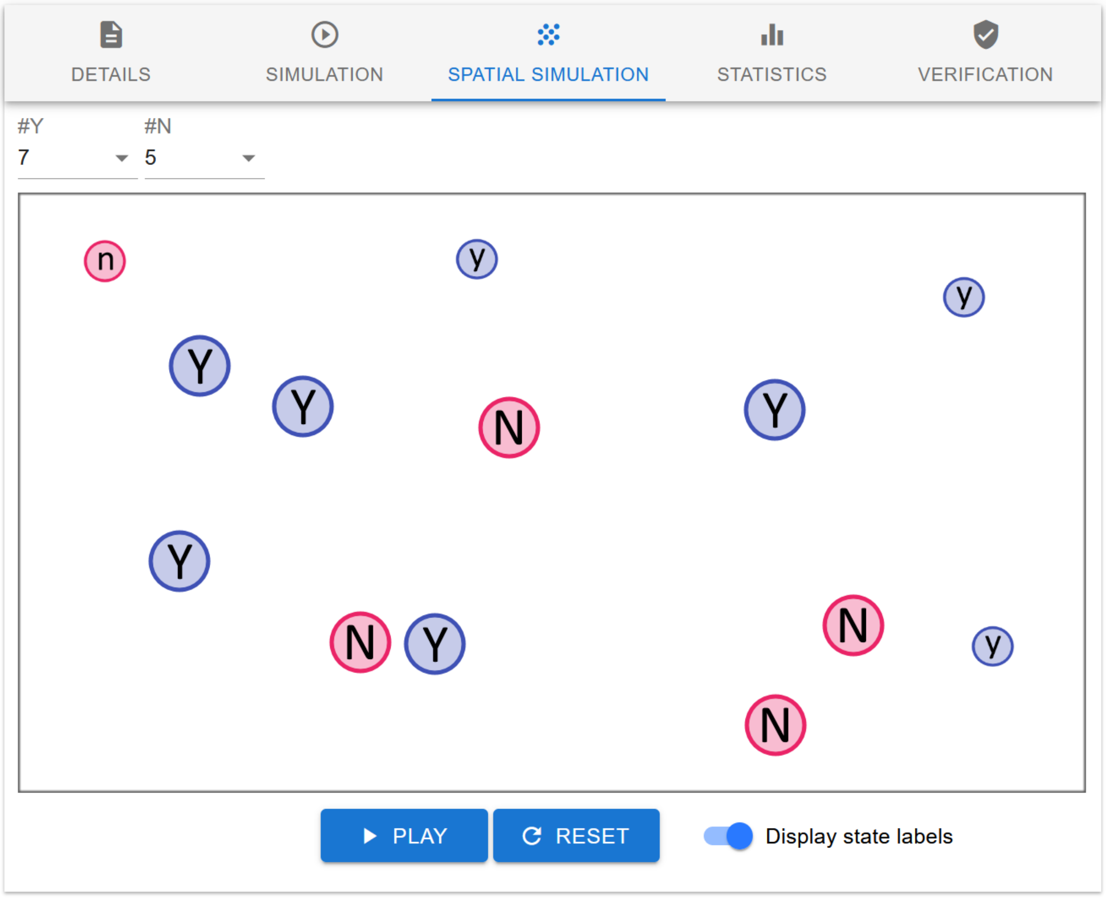


**Figure 3.5:** Spatial simulation of majority protocol (see Example 1) in tool Peregrine (see Section 4.3). Agents are drawn as molecules that randomly move in space. When two molecules collide, they change their state.

the transition is chosen with non-zero probability. (ii) If a configuration is infinitely often reachable in one step, it will almost surely be visited infinitely often because the corresponding transition is chosen with non-zero probability.

**Speed.** In the stochastic setting, it is possible to define the notion of time as the number of interactions that occur. The speed of a population protocol is then the expected number of interactions until the protocol stabilizes to the correct output. For example, the speed of the majority protocol of Example 1 is $2^{\mathcal{O}(n \log n)}$, where $n$ is the number of agents. The flock-of-birds protocol of Example 7 is much faster and needs only $\mathcal{O}(n^2)$ interactions to stabilize.

# 4 Verification of Population Protocols

When creating new population protocols or adapting existing ones, it is easy to make mistakes that result in incorrect protocols.[1] Consequently, it is very important to verify that a given protocol works as intended. While running simulations can increase confidence, this kind of testing does not guarantee correctness. The only way to be sure is to generate a proof for the protocol's correctness. As doing so by hand is time-consuming and error-prone, we investigate the *automatic verification* of population protocols.

**Definition 1** (Correctness Problem).
> ***Given:*** *A population protocol $\mathcal{P}$ and a formula $\varphi$.*
> ***Decide:*** *Does $\mathcal{P}$ compute $\varphi$?*

Note that for a *fixed* input, one can easily check if a population protocol computes the correct output by analyzing the finite reachability graph (see Example 3). However, the correctness problem is a *parameterized* verification problem as population protocols must be correct for all the *infinitely many* initial configurations. Although the correctness problem is still decidable, it is as hard as the reachability problem for vector addition systems [EGL+17], which is Ackermann-complete [CO22; Ler22].

This chapter has three parts: In Section 4.1, we introduce formal objects called stage graphs and show that they are a sound and complete technique for the verification of population protocols. Next, we give a practical algorithm for the efficient verification of population protocols in Section 4.2. Finally, in Section 4.3, we present an easy-to-use tool for the verification of population protocols, which implements this algorithm.

## 4.1 Theory of Stage Graphs

We will motivate our approach by analyzing the following correctness proof of the majority protocol of Example 1 that computes the formula $Y \geq N$ and has the four transitions:

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad t_4 : y, n \mapsto y, y$$

**Example 8** (Correctness Proof for Majority Protocol). *Agents in initial states ($Y$ and $N$) are called active. Notice that no transition increases the number of active agents and the transition $t_1$ reduces their number. Thus, any fair execution will eventually reduce the number of active agents until $t_1$ becomes disabled, Form this point on, it holds that either $Y = 0$ or $N = 0$. Then there are two cases: (a) In case the formula was false (i.e., $Y < N$), there is no agent in*

[1]For an example see [AGV15] where a typo in Figure 1 resulted in an incorrect protocol.

*$Y$ but at least one agent in $N$. From any reachable configuration, this agent can convert all $y$ into $n$ using transition $t_3$. Thus, any fair execution will eventually reach a configuration with consensus* false. *This is a stable consensus, as all non-silent transitions are disabled. (b) In cases where the formula was true (i.e., $Y \geq N$), there is no agent in $N$. Thus, the number of agents in $n$ must decrease with every non-silent transition. Further, there must be at least one agent in $y$ because the last occurrence of $t_1$ created one. Thus, any fair execution will eventually reach a configuration with consensus* true. *This is a stable consensus, as all non-silent transitions are disabled.*

Notice that the correctness proof has three phases. In the first phase, the protocol can use all transitions and the number of reachable configurations is large. In the second phase, the transition $t_1$ can never be used again, effectively limiting the number of reachable configurations. Finally, in the third phase, only silent transitions can occur, i.e., the number of reachable configurations is one. Indeed, any correct population protocol stabilizes to the correct output at which point many configurations, such as those with no or incorrect output, remain unreachable. This hints at a fundamental property of executions in correct population protocols: they get trapped in increasingly constrained sets of configurations. We formalize this idea by introducing *stage graphs* that can act as certificates for the correctness of population protocols.

### 4.1.1 Stable Termination

Stage graphs verify not only correctness but a more general class of *stable termination properties*. A stable termination property for a population protocol with states $Q$ is a pair $\Pi = (\varphi_{\text{pre}}, \Phi_{\text{post}})$ of a precondition $\varphi_{\text{pre}}$ and a set of postconditions $\Phi_{\text{post}} = \{\varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}\}$ where $\varphi_{\text{pre}}, \varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}$ are Presburger formulas over $Q$. When there is a single postcondition (i.e., $k = 1$), we also write $\Pi = (\varphi_{\text{pre}}, \varphi_{\text{post}})$. The pair $\Pi$ induces the *LTL* formula:

$$\varphi_{\text{pre}} \implies F \bigvee_{0 \leq i \leq n} G\varphi^i_{\text{post}}$$

In words: If the precondition holds, then eventually some postcondition holds forever.

**Definition 2** (Stable Termination Problem)**.**
> ***Given:*** *A population protocol $\mathcal{P}$ and a stable termination property $\Pi$.*
> ***Decide:*** *Does $\mathcal{P}$ satisfy $\Pi$?*

Recall the definition of computation for population protocols:

$$(\varphi_{\text{init}} \land \varphi \implies \mathbf{FG}\varphi^{\mathit{true}}_{\text{cons}}) \land (\varphi_{\text{init}} \land \neg\varphi \implies \mathbf{FG}\varphi^{\mathit{false}}_{\text{cons}})$$

As we can see, a protocol $\mathcal{P}$ computes a formula if and only if it satisfies the two stable termination properties $\Pi_{\mathit{true}} = (\varphi_{\text{init}} \land \varphi, \varphi^{\mathit{true}}_{\text{cons}})$ and $\Pi_{\mathit{false}} = (\varphi_{\text{init}} \land \neg\varphi, \varphi^{\mathit{false}}_{\text{cons}})$.

**Theorem 1.** *The correctness problem is polynomially reducible to the stable termination problem.*

### 4.1.2 Stage Graphs

To introduce stage graphs, we need the following definitions:

**Definition 3** (Inductive Set, Leads to, Certificate)**.**

- *A set of configurations is* inductive *if $C \in \mathscr{C}$ and $C \rightarrow C'$ implies $C' \in \mathscr{C}$.*
- *When $\mathscr{C}, \mathscr{C}'$ are sets of configurations, we say that $\mathscr{C}$* leads to *$\mathscr{C}'$, denoted as $\mathscr{C} \rightsquigarrow \mathscr{C}'$, if every fair run starting in $\mathscr{C}$ eventually visits a configuration of $\mathscr{C}'$.*
- *A* certificate *for $\mathscr{C} \rightsquigarrow \mathscr{C}'$ is a function $f : \mathscr{C} \mapsto \mathbb{N}$ such that for every configuration $C \in \mathscr{C} \setminus \mathscr{C}'$ there is a configuration $C'$ with $C \xrightarrow{*} C'$ and $f(C) > f(C')$.*

We will now argue that we can use a certificate $f$ to prove that an inductive set $\mathscr{C}$ leads to another set $\mathscr{C}'$. Assume there is a fair execution $\pi$ that does not visit $\mathscr{C}'$. As the number of reachable configurations is finite, but executions are infinite, $\pi$ has configurations that are visited infinitely often. Let C be a configuration with minimal certificate value among the infinitely visited configurations. As $\mathscr{C}$ is inductive, it must hold that $C \in \mathscr{C} \setminus \mathscr{C}'$. Due to the certificate $f$, there is a configuration $C'$ that is reachable from $C$ with $f(C) > f(C')$. As $\pi$ is fair, it must also visit $C'$ infinitely often. However, this implies that $C$ was not the configuration with the lowest certificate value among all configurations that are visited infinitely often, contradicting our initial assumption.

**Proposition 1.** *If there is a certificate for $\mathscr{C} \rightsquigarrow \mathscr{C}'$ and $\mathscr{C}$ is inductive, then $\mathscr{C} \rightsquigarrow \mathscr{C}'$.*

Now we can define stage graphs:

**Definition 4** (Stage Graph)**.** *A* stage graph *for stable termination property $\Pi$ is an acyclic graph. Its nodes are sets of configurations, called* stages*. A stage is* terminal *if is has no successors. Otherwise, it is* non-terminal*. A stage graph satisfies the following four properties:*

*(1) Every stage is inductive.*

*(2) Every configuration that satisfies the precondition $\varphi_{\text{pre}}$ is in some stage.*

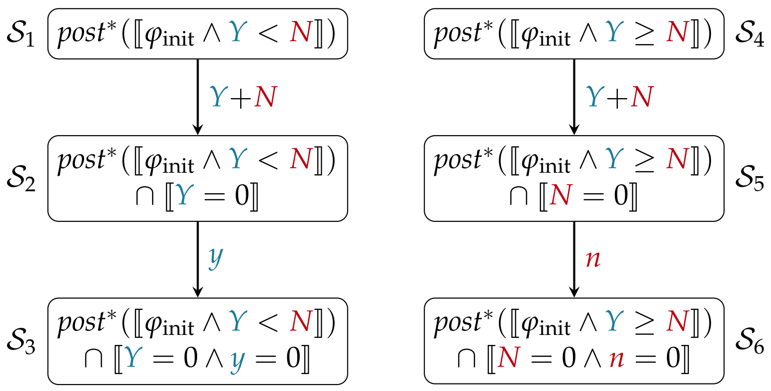
*(3) If $\mathcal{S}$ is a non-terminal stage with successors $\mathcal{S}_1, \ldots, \mathcal{S}_n$, then there exists a certificate for $\mathcal{S} \rightsquigarrow (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_n)$.*

*(4) If $\mathcal{S}$ is a terminal stage, then $\mathcal{S} \subseteq [\![\varphi^i_{\text{post}}]\!]$ for some $i$.*

We will now argue that a stage graph for stable termination property $\Pi$ proves that the protocol satisfies $\Pi$. A fair execution that starts in a configuration satisfying the precondition $\varphi_{\text{pre}}$ also starts in a stage by property (2). As long as the fair execution is not in a terminal stage, it must eventually enter a successor because of the certificate in property (3) and Proposition 1. This continues until the execution enters a terminal stage that it can never leave because of property (1). From this point on, every configuration satisfies postcondition $\varphi^i_{\text{post}}$ by property (4).

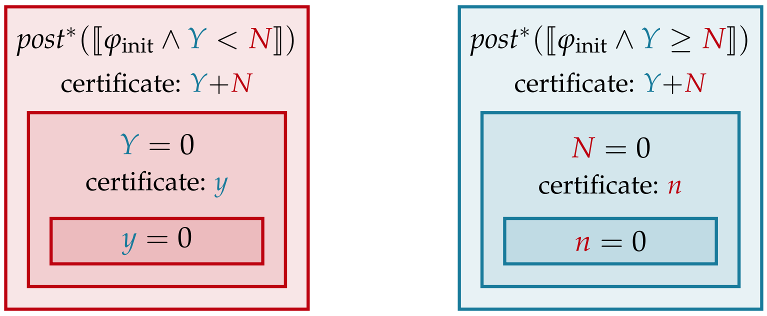
**Theorem 2.** *If a population protocol $\mathcal{P}$ has a stage graph for a stable termination property $\Pi$, then $\mathcal{P}$ satisfies $\Pi$.*

**Example 9** (Stage Graphs for Correctness of Majority Protocol)**.** *Figure 4.1 shows a pair of stage graphs that verify the correctness of the majority protocol defined in Example 1. Certificates are written on the edges. Note that the stage graphs closely correspond to the correctness proof given in Example 8. Often it is helpful to visualize stage graphs as* Venn diagrams *[Ven80] where configurations are points on the plane and sets of configurations are regions. For example, see Figure 4.2, which shows how the majority protocol becomes trapped in increasingly constrained sets of configurations.*



$\mathcal{S}_1$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y < N \rrbracket)$
$Y+N$
$\mathcal{S}_2$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y < N \rrbracket) \cap \llbracket Y = 0 \rrbracket$
$y$
$\mathcal{S}_3$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y < N \rrbracket) \cap \llbracket Y = 0 \wedge y = 0 \rrbracket$
$\mathcal{S}_4$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y \geq N \rrbracket)$
$Y+N$
$\mathcal{S}_5$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y \geq N \rrbracket) \cap \llbracket N = 0 \rrbracket$
$n$
$\mathcal{S}_6$ $post^*(\llbracket \varphi_{\text{init}} \wedge Y \geq N \rrbracket) \cap \llbracket N = 0 \wedge n = 0 \rrbracket$

**Figure 4.1:** Stage graphs verifying the correctness of the majority protocol defined in Example 1. The left stage graph verifies that the protocol stabilizes to *false* if there was a majority for $N$ in the initial configuration, and the right stage graph verifies the other case. See Figure 4.2 for a Venn-diagram visualization of the same stage graphs.



$post^*(\llbracket \varphi_{\text{init}} \wedge Y < N \rrbracket)$
certificate: $Y+N$
$Y = 0$
certificate: $y$
$y = 0$
$post^*(\llbracket \varphi_{\text{init}} \wedge Y \geq N \rrbracket)$
certificate: $Y+N$
$N = 0$
certificate: $n$
$n = 0$

**Figure 4.2:** Venn-diagram visualization of stage graphs in Figure 4.1 highlighting that the majority protocol becomes trapped in increasingly constrained sets of configurations.

**Stage Graphs Require Global Fairness.** Stage graphs verify stable termination under the assumption of *global* fairness. If we use *local* fairness instead, a certificate does not guarantee that executions eventually enter a successor. An example for this is given in Figure 4.3.

$$\begin{aligned} Q &\overset{\text{def}}{=} \{A, B, \bot\} \\ T &\overset{\text{def}}{=} \{t_A : A, B \mapsto A, A \\ &\qquad t_B : A, B \mapsto B, B \\ &\qquad t_\bot : A, A \mapsto \bot, \bot\} \end{aligned}$$

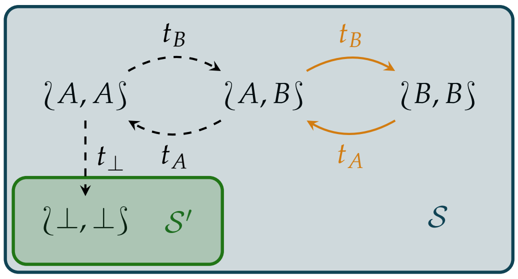


**Figure 4.3:** Certificates require global fairness. Consider the following two stages of the population protocol on the left: $\mathcal{S} = \{\Lbag A, A \Rbag, \Lbag A, B \Rbag, \Lbag B, B \Rbag, \Lbag \bot, \bot \Rbag\}$ and its successor $\mathcal{S}' = \{\Lbag \bot, \bot \Rbag\}$. A certificate for $\mathcal{S} \rightsquigarrow \mathcal{S}'$ is $f(C) \overset{\text{def}}{=} C(A)$. As configuration $\Lbag \bot, \bot \Rbag$ is always reachable, a globally fair execution cannot avoid it forever. However, a locally fair execution, such as the one drawn as orange cycle in the reachability graph on the right, can. This execution is locally fair because only $t_A$ and $t_B$ are enabled infinitely often and both are used infinitely often.

## Stage Graphs are Sound and Complete

So far, we argued that the existence of a stage graph can certify a stable termination property. In addition, we can also show that if some stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{\text{post}})$ with $\Phi_{\text{post}} = \{\varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}\}$ holds, then there is always a stage graph that certifies this fact. Consider a stage graph with $k + 1$ stages: one initial stage $\mathcal{S}_{init} \overset{\text{def}}{=} post^*(\llbracket \varphi_{\text{pre}} \rrbracket)$ containing all configurations reachable from configurations satisfying the precondition, and $k$ terminal successor stages $\mathcal{S}_i \overset{\text{def}}{=} \llbracket \mathbf{G} \varphi^i_{\text{post}} \rrbracket$ containing all configurations that satisfy postcondition $\varphi_{\text{post}}$ forever (see Figure 4.4).

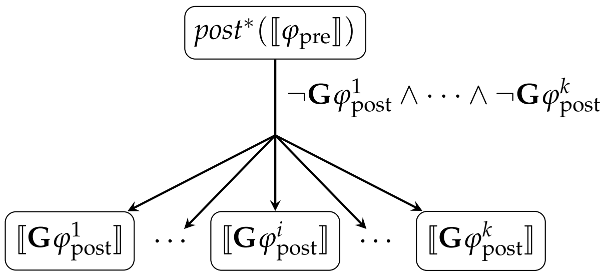

$post^*(\llbracket \varphi_{\text{pre}} \rrbracket)$
$\neg \mathbf{G} \varphi^1_{\text{post}} \wedge \cdots \wedge \neg \mathbf{G} \varphi^k_{\text{post}}$
$\llbracket \mathbf{G} \varphi^1_{\text{post}} \rrbracket$
$\cdots$
$\llbracket \mathbf{G} \varphi^i_{\text{post}} \rrbracket$
$\cdots$
$\llbracket \mathbf{G} \varphi^k_{\text{post}} \rrbracket$

**Figure 4.4:** Stage graph constructed to prove Theorem 3. If a protocol satisfies the stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{\text{post}})$ with $\Phi_{\text{post}} = \{\varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}\}$, then the given graph is a stage graph for $\Pi$.

Properties (1), (2), and (4) of stage graphs hold by definition. Since $\mathcal{P}$ satisfies $\Pi$, every fair execution starting in a configuration satisfying the precondition eventually enters a terminal stage, i.e., $\mathcal{S}_{init} \rightsquigarrow (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$. Thus, the function $f(C) \overset{\text{def}}{=} C \in (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$ is a certificate for property (3). We conclude:

**Theorem 3.** *If a population protocol $\mathcal{P}$ satisfies a stable termination property $\Pi$, then $\mathcal{P}$ has a stage graph for $\Pi$.*

By combining Theorems 2 and 3, we know that stage graphs are a sound and complete method for proving stable termination.

**Theorem 4.** *A population protocol $\mathcal{P}$ satisfies a stable termination property $\Pi$ if and only if it has a stage graph for $\Pi$.*

### 4.1.3 Presburger Stage Graphs

Notice that the stages and certificates of stage graphs can be arbitrarily complicated. Thus, Theorem 4 does not prove that the stable termination problem (and by extension the correctness problem) is decidable. However, we can strengthen the result by only considering "simple" *stage graphs* where (i) stages and certificates are expressed by Presburger formulas and (ii) certificate values can be reduced in a constant number of interactions.

**Definition 5** (Bounded Certificate)**.** *A* bounded certificate *for $\mathscr{C} \rightsquigarrow \mathscr{C}'$ is a pair $(f, k)$ of a function $f : \mathscr{C} \mapsto \mathbb{N}$ and bound $k \in \mathbb{N}$, such that for every configuration $C \in \mathscr{C} \setminus \mathscr{C}'$ there is a transition sequence $w = t_0, t_1, \ldots, t_j$ with $j \leq k$ such that $C \xrightarrow{w} C'$ and $f(C) > f(C')$.*

Intuitively, a bounded certificate is easy to check because it guarantees that its value can be reduced with a short transition sequence instead of an arbitrary sequence.

**Definition 6** (Presburger Stage Graph)**.** *A* Presburger stage graph *is a stage graph such that*

- *stages are Presburger sets, i.e., for every stage $\mathcal{S}$ there is a Presburger formula $\varphi$ such that $C \in \mathcal{S} \Leftrightarrow \varphi(C)$, and*
- *certificates are bounded and given as Presburger formulas, i.e., for every bounded certificate $(f, k)$ there is a Presburger formula $\varphi$ such that $f(C) = n \Leftrightarrow \varphi(C, n)$.*

Note that the stage graphs in Example 9 are Presburger stage graphs.

#### Presburger Stage Graphs are Sound and Complete

We will now argue that even with this limitation, stage graphs are still a sound and complete technique for the verification of stable termination properties. It is easy to see that a Presburger stage graph is still a stage graph, and thus Presburger stage graphs are still sound. The proof for completeness first assumes that a population protocol $\mathcal{P}$

satisfies a stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{\text{post}})$ with $\Phi_{\text{post}} = \{\varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}\}$. The rest of the proof has two parts: (a) argue that there is a stage graph where all stages are Presburger sets, and (b) argue that all certificates are Presburger certificates. Because the second part is more technical and uses deep results from the theory of Petri nets, we will focus on the first part.

The Presburger stage graph proving $\Pi$ has $k$ terminal stages (one for each postcondition) and one non-terminal stage. It is depicted in Figure 4.5. Additionally, Figure 4.6 shows a Venn diagram of the involved sets of configurations and highlights which of the sets are Presburger.

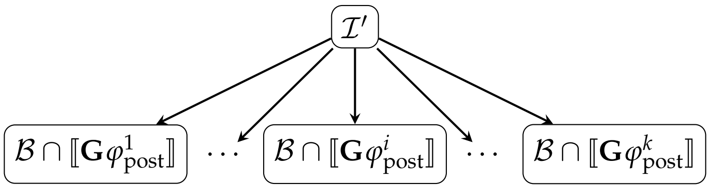

$\mathcal{I}'$
$\mathcal{B} \cap [\![\mathbf{G}\varphi^1_{\text{post}}]\!]$
$\cdots$
$\mathcal{B} \cap [\![\mathbf{G}\varphi^i_{\text{post}}]\!]$
$\cdots$
$\mathcal{B} \cap [\![\mathbf{G}\varphi^k_{\text{post}}]\!]$

**Figure 4.5:** Presburger stage graph constructed to prove completeness of Presburger stage graphs for Theorem 5. If a protocol satisfies the stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{\text{post}})$ with $\Phi_{\text{post}} = \{\varphi^1_{\text{post}}, \dots, \varphi^k_{\text{post}}\}$, then the given graph is a Presburger stage graph for $\Pi$. The set $\mathcal{I}'$ is a Presburger overapproximation of $post^*([\![\varphi_{\text{pre}}]\!])$. See Figure 4.6 for a Venn-diagram visualization of the involved sets of configurations.

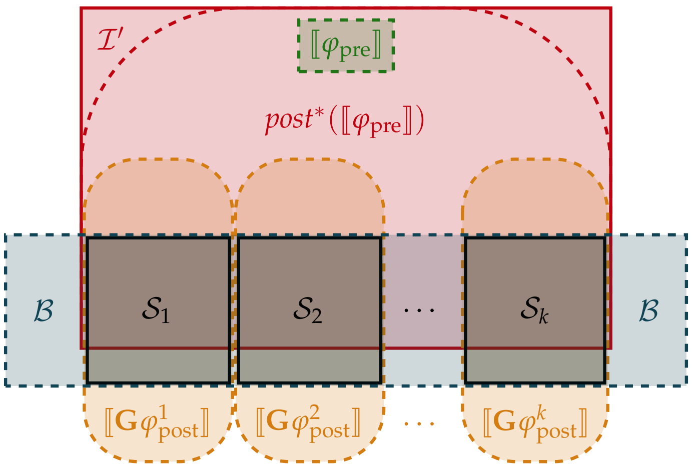

$\mathcal{I}'$
$[\![\varphi_{\text{pre}}]\!]$
$post^*([\![\varphi_{\text{pre}}]\!])$
$\mathcal{B}$
$\mathcal{S}_1$
$\mathcal{S}_2$
$\cdots$
$\mathcal{S}_k$
$\mathcal{B}$
$[\![\mathbf{G}\varphi^1_{\text{post}}]\!]$
$[\![\mathbf{G}\varphi^2_{\text{post}}]\!]$
$\cdots$
$[\![\mathbf{G}\varphi^k_{\text{post}}]\!]$

**Figure 4.6:** Venn-diagram visualization for sets of configurations involved in the Presburger stage graph of Figure 4.5. Stages have solid borders while all other sets have dashed borders. For Presburger sets, the borders consist of straight lines. For non-Presburger sets, they are curved.

**Terminal Stages.** We use the following idea for the terminal stages of the Presburger stage graph: As the reachability graph of a population protocol is always finite, any fair execution of the system will eventually enter one of the *bottom strongly connected components* of the reachability graph. Intuitively, at this point the system is "maximally trapped" as all reachable configurations remain reachable. Formally, we say that a configuration is *bottom* if $C \xrightarrow{*} D$ implies $D \xrightarrow{*} C$. Let $\mathcal{B}$ be the set of all bottom configurations. The terminal stages of the Presburger stage graph will be $\mathcal{S}_i \overset{\text{def}}{=} \mathcal{B} \cap [\![\mathbf{G}\varphi^i_{\text{post}}]\!]$ for all $1 \leq i \leq k$. I.e., $\mathcal{S}_i$ are the bottom configurations that continuously satisfy postcondition $i$. To understand why the terminal stages are Presburger sets, we need the following two results from [EGL+17]:

**Proposition 2** (adapted from [EGL+17, Thm. 13]; original result in [ES69])**.** *The mutual reachability relation defined as $C \xleftrightarrow{*} D \overset{def}{\iff} (C \xrightarrow{*} D \wedge D \xrightarrow{*} C)$ is Presburger definable.*

**Proposition 3** (adapted from [EGL+17, Prop. 14])**.** *$\mathcal{B}$ is a Presburger set.*

Note that reachability and mutual reachability coincide for bottom configurations. Thus, we can give the following Presburger formula for $C \in \mathcal{S}_i$:

$$C \in \mathcal{B} \wedge \forall D : C \xleftrightarrow{*} D \implies \varphi^i_{\text{post}}(D)$$

Intuitively, these are the bottom configurations that can reach only configurations that satisfy postcondition $i$.

**Non-Terminal Stages.** Stage graph property (2) requires that the non-terminal stage contains $[\![\varphi_{\text{pre}}]\!]$. Ideally, we would like to use the stage $\mathcal{I} = post^*([\![\varphi_{\text{pre}}]\!])$ just as in the non-Presburger construction. Because any execution eventually enters $\mathcal{B}$ and the stable termination property holds, we know $\mathcal{I} \rightsquigarrow (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$. However, $\mathcal{I}$ is not Presburger, and thus we use an inductive Presburger overapproximation $\mathcal{I}' \supseteq \mathcal{I}$ as non-terminal stage. If $\mathcal{I}'$ does not contain bottom configurations that are in no terminal stage, then $\mathcal{I}' \rightsquigarrow (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$ by the same reasoning. Such an overapproximation exists because of the following result from [Ler12]:

**Proposition 4** (weakened version of [Ler12, Lem. 9.1])**.** *For Presburger sets of configurations $\mathscr{C}, \mathscr{D} \in \mathbb{N}^Q$ with $post^*(\mathscr{C}) \cap \mathscr{D} = \emptyset$, there is an inductive Presburger set $\mathcal{I}' \supseteq \mathscr{C}$ with $\mathcal{I}' \cap \mathscr{D} = \emptyset$.*

Intuitively, Proposition 4 tells us that if $\mathscr{C}$ cannot reach $\mathscr{D}$, then there is a Presburger "barrier" between the two sets. In our case, we use $\mathscr{C} \overset{\text{def}}{=} [\![\varphi_{\text{pre}}]\!]$ and $\mathscr{D} \overset{\text{def}}{=} \mathcal{B} \setminus (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$. As $\mathscr{C}$ cannot reach $\mathscr{D}$ because the stable termination property holds, this yields the needed non-terminal stage $\mathcal{I}'$ that contains $[\![\varphi_{\text{pre}}]\!]$ with $\mathcal{I}' \rightsquigarrow (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k)$.

**Theorem 5.** *A population protocol $\mathcal{P}$ satisfies a stable termination property $\Pi$ if and only if it has a Presburger stage graph for $\Pi$.*

### 4.1.4 Stable Termination is Decidable

Observe that we can check if a given graph is a Presburger stage graph by checking the satisfiability of multiple Presburger formulas. For example, to check that a stage $\mathcal{S}$ is inductive, we can show that there is no configuration $C \in \mathcal{S}$ that has an enabled transition $t$ that leads out of the stage. This can be achieved by showing that the following Presburger formula is unsatisfiable:

$$\exists C \in \mathcal{S} : \bigvee_{t \in T} (C \geq {}^\bullet t \wedge (C + \Delta(t)) \notin \mathcal{S})$$

**Theorem 6.** *Deciding whether an acyclic graph of stages (given as Presburger formulas) and certificates (given as Presburger functions) is a Presburger stage graph for a stable termination property is polynomially reducible to the satisfiability problem of Presburger arithmetic.*

It is known that the satisfiability problem of Presburger arithmetic is decidable [Pre29] and has complexity between 2-NEXP and 2-EXPSPACE [Ber80]. Thus, we can semi-decide the stable termination problem by enumerating all possible Presburger stage graphs and checking if they prove the property.

To decide stable termination, we run a second search in parallel until we find a counterexample for stable termination. It enumerates all configurations satisfying the precondition, builds the finite reachability graph, and then checks for each bottom strongly connected component that all configurations satisfy a common postcondition.

**Theorem 7.** *The stable termination problem for population protocols is decidable.*

Combined with Theorem 1, we get an alternative proof for the decidability of the correctness problem given in [EGL+17].

**Corollary 7.1.** *The correctness problem for population protocols is decidable.*

## 4.2 Efficient Construction of Stage Graphs

Our goal is the automatic verification of stable termination properties. We could use the procedure of the decidability result in Theorem 7 and simply enumerate Presburger stage graphs until we find one that proves the property. However, this approach has two major problems:

1. From a theoretical point of view, the Ackerman-hardness of the correctness problem[2] implies that the smallest Presburger stage graph can be very large in terms of the system's size.
2. In practice, the enumeration of all candidates is infeasible even though most systems have reasonably small stage graphs with just 2-5 stages.

Therefore, we introduce an algorithm that automatically *constructs* a Presburger stage graph using a series of heuristics. It makes use of a constraint solver for the existential fragment of Presburger arithmetic to search for certain kinds of stage graphs where stages correspond to the "death" of transitions.

**Definition 7** (Death of Transitions)**.**

- *A transition $t$ is* dead *in configuration $C$ if it can never occur in any execution starting at $C$, i.e., if $C$ satisfies the formula* $\mathrm{Dead}(t) \overset{def}{=} \mathbf{G}\neg(C \geq {}^\bullet t)$.
- *A set of transitions $U \subseteq T$ is dead in configuration $C$ if all transitions are dead, i.e., if $C$ satisfies the formula* $\mathrm{Dead}(U) \overset{def}{=} \bigwedge_{t \in U} \mathrm{Dead}(t)$.

As before, this notion can be lifted to sets of configurations, e.g., the transitions $U \subseteq T$ are dead in a set of configurations $\mathscr{C}$ if $\mathscr{C} \subseteq [\![ Dead(U) ]\!]$. As we limit our search for stage graphs, our procedure may fail, but crucially, it works for most systems in the literature.

### Stage Representation

Before we explain the efficient algorithm for constructing stage graphs, we will first describe how the stages are defined and represented. For this, we need to introduce the notion of upward closed sets.

**Definition 8** (Upward Closed Set)**.** *Let $\mathscr{C} \subseteq \mathbb{N}^Q$ be a set of configurations. The* upward closure *of a set of configurations $\mathscr{C}$ is $\uparrow\mathscr{C} \overset{def}{=} \{C \in \mathbb{N}^Q \mid C \geq C' \text{ for some } C' \in \mathscr{C}\}$, i.e., the configurations that are larger than (or equal to) at least one of the configurations in $\mathscr{C}$. Special cases are $\uparrow\{\{\!\!\}\} = \mathbb{N}^Q$ and $\uparrow\emptyset = \emptyset$. We say $\mathscr{C}$ is* upward closed *if $\mathscr{C} = \uparrow\mathscr{C}$. The* basis *of an upward closed set $\mathscr{C}$ is the minimal set of configurations $\inf(\mathscr{C})$ such that $\uparrow\inf(\mathscr{C}) = \mathscr{C}$, i.e., the basis contains exactly the minimal configurations of the upward closed set.*

[2]The reachability problem for vector addition systems, which is Ackermann-complete [CO22; Ler22], can be polynomially reduced to the correctness problem [EGL+17].

**Example 10.** *The set* $\mathscr{C} \overset{def}{=} [\![(A \geq 1 \wedge B \geq 2) \vee C \geq 1]\!]$ *is upward closed and* $\mathscr{C}$ *is the upward closure of both* $\mathscr{D} \overset{def}{=} \{\wr A, 2 \cdot B \wr, \wr C \wr\}$ *and* $\mathscr{E} \overset{def}{=} \{\wr A, 2 \cdot B \wr, \wr C \wr, \wr 2 \cdot C \wr\}$*. However, only* $\mathscr{D}$ *is the basis of* $\mathscr{C}$ *as* $\mathscr{E}$ *contains the redundant configuration* $\wr 2 \cdot C \wr$ *that is already part of the upward closure because of* $\wr C \wr$*.*

For the stage representation, we leverage the following property that emerges naturally from the definitions of upward-closed sets and $Dead(U)$:

**Proposition 5.** *The set* $[\![\neg \mathrm{Dead}(U)]\!]$ *for a set of transitions* $U \subseteq T$ *is upward-closed.*

Intuitively, the set of configurations that can enable a transition, i.e., the configurations where a transition is not dead, is upward-closed.

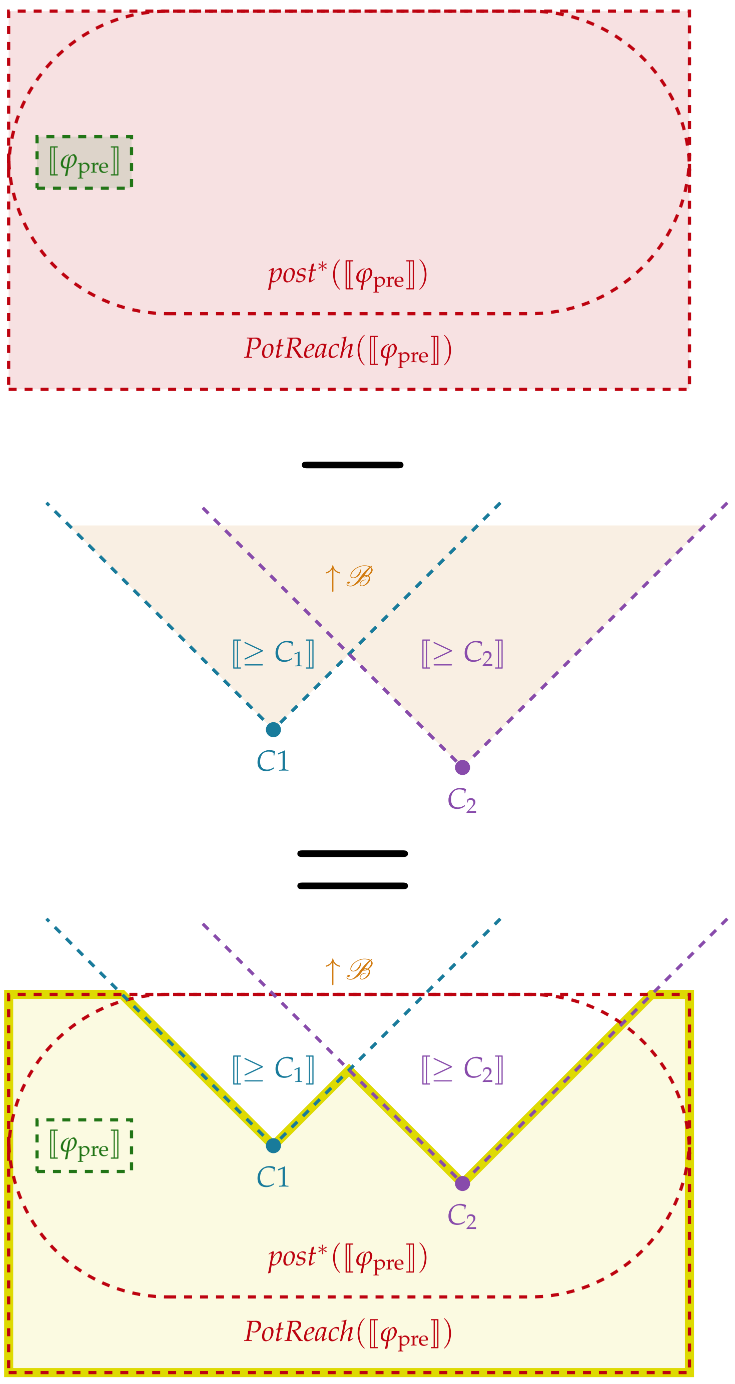
The stages produced by our algorithm have a specific form: Each stage is represented by a pair $(D, \mathscr{B})$ where $D \subseteq T$ are the dead transitions of the stage and $\mathscr{B} \subseteq \mathbb{N}^Q$ is a minimal basis of an upward closed set we use to exclude the configurations $\uparrow \mathscr{B}$. Further, we demand that all those configurations where $D$ is not dead are excluded, i.e., we force $[\![\neg Dead(U)]\!] \subseteq \uparrow \mathscr{B}$. The configurations in stage $(D, \mathscr{B})$ are $PotReach([\![\varphi_{\mathrm{pre}}]\!]) \cap \overline{\uparrow \mathscr{B}}$, where *PotReach* is an inductive Presburger overapproximation of the non-Presburger $post^*$. More details about *PotReach* are given in Section 4.2.1. See Figure 4.7 for a Venn-diagram visualization of the stage definition.

**Example 11.** *To show the stable termination property* $\Pi_{\mathrm{true}} = (\varphi_{\mathrm{init}} \wedge Y \geq N, N = 0 \wedge n = 0)$ *for the majority voting protocol (see Example 1), all stages have the form* $\mathrm{PotReach}([\![\varphi_{\mathrm{init}} \wedge Y \geq N]\!]) \cap \overline{\uparrow \mathscr{B}}$*. Intuitively, they contain all those configurations that (i) are (potentially) reachable from some initial configuration with majority for* $Y$*, and (ii) are not in the upward closed set* $\uparrow \mathscr{B}$*. In a stage with dead transitions* $D = \{t_1 : Y, N \mapsto y, n\}$*, we must exclude at least those transitions where* $t_1$ *is not dead. This can be achieved with basis* $\mathscr{B} = \{\wr Y, N \wr\}$ *as this excludes all configurations where* $t_1$ *is enabled, and once* $t_1$ *is disabled it cannot become enabled again.*

## Algorithm

Our algorithm maintains a workset containing stages for which we still need to prove stable termination. Initially, the workset contains only one stage covering all configurations that satisfy the precondition, i.e., the stage $(\emptyset, \emptyset)$ where no transitions are dead and no configurations are excluded. While the workset is not empty, the algorithm always removes a stage $\mathcal{S}$ with representation $(D, \mathscr{B})$ and performs the following steps:

1. **Check Stable Termination:** Try to prove that the current stage is terminal, i.e., that $\mathcal{S} \subseteq [\![\mathbf{G}\varphi^i_{\mathrm{post}}]\!]$ for some $i$. If the stage is terminal, proceed with the next stage.
2. **Improve Stage Representation:** Try to find transitions that are already dead in all configurations of the stage. If we find dead transitions, add them to $D$ and update $\mathscr{B}$ to exclude more configurations.



$[\![\varphi_{\text{pre}}]\!]$
$post^*([\![\varphi_{\text{pre}}]\!])$
$PotReach([\![\varphi_{\text{pre}}]\!])$
$\uparrow\mathscr{B}$
$[\![\geq C_1]\!]$
$[\![\geq C_2]\!]$
C1
$C_2$
$\uparrow\mathscr{B}$
$[\![\geq C_1]\!]$
$[\![\geq C_2]\!]$
$[\![\varphi_{\text{pre}}]\!]$
C1
$C_2$
$post^*([\![\varphi_{\text{pre}}]\!])$
$PotReach([\![\varphi_{\text{pre}}]\!])$

**Figure 4.7:** Venn-diagram visualization of automatically constructed stages. For Presburger sets, the borders consist of straight lines. For non-Presburger sets, they are curved. The stage is the area with the solid yellow border and is the result of excluding the upward-closed set $\uparrow\mathscr{B}$ from the set $PotReach([\![\varphi_{\text{pre}}]\!])$ where $\mathscr{B} = \{C_1, C_2\}$.

3. **Search Eventually Dead Transitions:** Try to find a certificate proving that some non-empty set of transitions $U \subseteq T \setminus D$ becomes dead in every fair execution starting in $\mathcal{S}$. If successful, add a *single successor* with dead transitions $D \cup U$ (and updated basis) to the workset and proceed with the next stage.

4. **Search Split:** Try to split $\mathcal{S}$ into parts $\mathcal{S}_1, \ldots, \mathcal{S}_n$ such that $\mathcal{S}$ is completely covered (i.e., $\mathcal{S} = (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_n)$) and in every $\mathcal{S}_i$ some additional transition $t_i \in T \setminus D$ is dead. If successful, add *all successors* to the workset.[3] Otherwise, the algorithm fails.

The following sections give additional details: Section 4.2.1 explains the Presburger overapproximation of reachability used in the stage definition. Section 4.2.2 shows how to update the basis of a stage in order to exclude configurations that can enable supposedly dead transitions. Section 4.2.3 explains how to check stable termination in Step 1. Section 4.2.4 explains how to find already dead transitions in Step 2. Section 4.2.5 describes how to find transitions that eventually become dead and a corresponding certificate in Step 3. Section 4.2.6 demonstrates how to split a stage in Step 4. Finally, we explain how we use stage graphs to automatically analyze the speed of population protocols in Section 4.2.7.

### 4.2.1 Potential Reachability

In this section, we will explain the inductive Presburger overapproximation *PotReach* used in the stage definition. As reachability of a Presburger set is not Presburger, we use an inductive Presburger overapproximation that was introduced in [BEJ+17; ELM+14]. It combines two techniques familiar from the theory of Petri nets: the *flow equation* (also known as marking equation) that describes the commutative effect of all transitions as well as constraints for *traps* and *siphons*.

**Flow Equation.** If a configuration $C$ can reach a configuration $C'$, then there necessarily is a finite sequence of transitions $\tau = t_1 t_2 t_3 \ldots t_n$ that leads from $C$ to $C'$. The combined effect of $\tau$ must be the difference between $C$ and $C'$, i.e., it holds that

$$C + \sum_{t \in T} |\tau|_t \cdot \Delta(t) = C'$$

where $|\tau|_t$ is the number of occurrences of transition $t$ in $\tau$. Thus, if $C \xrightarrow{*} C'$, then it is always possible to find a multiset $x \in \mathbb{N}^T$ that is a solution for the flow equation:

$$C + \sum_{t \in T} x(t) \cdot \Delta(t) = C'$$

Note that this is expressible as an existential Presburger formula and results in an inductive overapproximation of reachability. Indeed, just because there is a solution of

[3]The certificate can be an arbitrary function like $f(C) = 0$ because splitting is essentially a case distinction.

the flow equation, it must not hold that $C \xrightarrow{*} C'$, as only the total effect of transitions is taken into account. It can still be the case that no sequence of transitions with that total effect is realizable. As an example, consider the population protocol with a single non-silent transition $A, C \mapsto B, C$. Clearly, configuration $\wr A \wr$ cannot reach $\wr B \wr$, but there is a solution for the flow equation.

**Traps and Siphons.** Because the flow equation on its own is not precise enough for our use case, we combine it with constraints for traps and siphons. A set of states $R \subseteq Q$ is a *$U$-trap* if for every transition $t \in U$ it holds that ${}^\bullet t \cap R \neq \emptyset \implies t^\bullet \cap R \neq \emptyset$. In other words, every transition in $U$ that consumes an agent in trap $R$ produces an agent in $R$. Effectively, this implies that non-empty traps stay non-empty if we only use transitions in $U$. Conversely, a set of states $R \subseteq Q$ is a *$U$-siphon* if for every transition $t \in U$ it holds that $t^\bullet \cap R \neq \emptyset \implies {}^\bullet t \cap R \neq \emptyset$. In other words, every transition in $U$ that produces an agent in siphon $R$ consumes an agent in $R$. Effectively, this implies that empty $U$-siphons stay empty if we only use transitions in $U$. Note that by definition, the union of two $U$-siphons is a $U$-siphon and the union of two $U$-traps is a $U$-trap. Thus, every subset of states has a unique largest $U$-siphon and a unique largest $U$-trap.

**Combination.** Let $x$ be the solution to the flow equation and let $\tau$ be some corresponding transition sequence with $|\tau|_t = x(t)$ for all $t \in T$. Clearly, $\tau$ uses only the transitions $U = \{t \in T \mid x(t) > 0\}$. If $\tau$ leads from C to C′, then we know:

- Every empty $U$-siphon in C must still be empty in C′.
- Every empty $U$-trap in C′ must have been empty in C.

This allows us to show that some configurations are not reachable, even if there is a solution for the flow equation.

**Example 12** (Proving Unreachability using Siphons)**.** *In the population protocol of Figure 4.8, it is not possible to reach $C' = \wr B, C, D \wr$ from $C = \wr A, C, E \wr$. The agent in state $A$ must change to B, but this requires that there is already an agent in D. The only agent that can change to D is the agent in E, but this requires that there is already an agent in B. Despite this fact, there is a solution for the flow equation that uses both $t_1$ and $t_3$ once. (In fact, this is the only solution.) It is possible to prove that C cannot reach C′ using a siphon: The set $\{B, D\}$ is a $\{t_1, t_3\}$-siphon that is initially empty in C but not empty in C′. Note that the usage of the flow equation is necessary to prove unreachability: If we did not know that $t_2$ cannot occur, then the largest empty siphon in C is the empty set.*

**Example 13** (Proving Unreachability using Traps)**.** *In the population protocol of Figure 4.9 it is not possible to reach $C' = \wr B, C, E \wr$ from $C = \wr A, D, E \wr$. The agent in state A must change to B, but this requires that there is still an agent in D. The only agent that can change to C is the agent in D, but this requires that there is still an agent in A. Despite this fact, there is a solution for the flow equation that fires both $t_1$ and $t_2$ once. (In fact, this is the only solution.) It is possible to prove that C cannot reach C′ using a trap: The set $\{A, D\}$ is a $\{t_1, t_2\}$-trap that is*

$$\begin{aligned} Q &\overset{\text{def}}{=} \{A, B, C, D, E\} \\ T &\overset{\text{def}}{=} \{t_1 : A, D \mapsto B, D \\ &\qquad t_2 : A, C \mapsto A, D \\ &\qquad t_3 : B, E \mapsto B, D\} \\ C &\overset{\text{def}}{=} \Lbag A, C, E \Rbag \\ C' &\overset{\text{def}}{=} \Lbag B, C, D \Rbag \end{aligned}$$

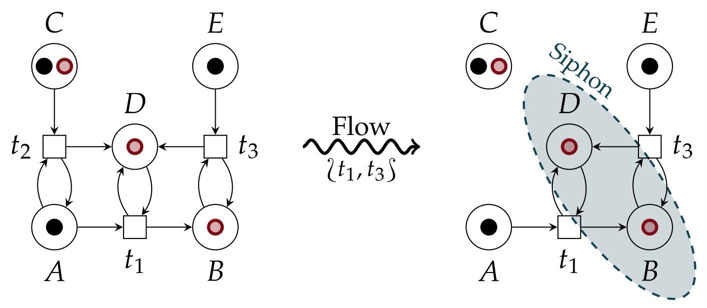


**Figure 4.8:** Illustration of Example 4.8 showing how to prove that configuration $C$ cannot reach configuration $C'$. The population protocol is formally defined (left) and visualized as a Petri net (middle). The two configurations are drawn using tokens of different colors. When considering only the transitions that occur in the solution to the flow equation, there is a siphon that is empty in $C$ but non-empty in $C'$ (right).

$$\begin{aligned} Q &\overset{\text{def}}{=} \{A, B, C, D, E\} \\ T &\overset{\text{def}}{=} \{t_1 : A, D \mapsto B, D \\ &\qquad t_2 : A, D \mapsto A, C \\ &\qquad t_3 : B, D \mapsto B, E\} \\ C &\overset{\text{def}}{=} \Lbag A, D, E \Rbag \\ C' &\overset{\text{def}}{=} \Lbag B, C, E \Rbag \end{aligned}$$

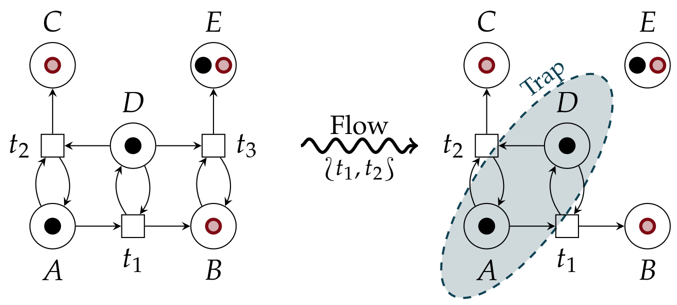


**Figure 4.9:** Illustration of Example 4.9 showing how to prove that configuration $C$ cannot reach configuration $C'$. The population protocol is formally defined (left) and visualized as Petri net (middle). The two configurations are drawn using tokens of different colors. When considering only the transitions that occur in the solution to the flow equation, there is a trap that is empty in $C'$ but non-empty at $C$ (right).

*empty in $C'$ but not empty in $C$. Note that the usage of the flow equation is necessary to prove unreachability: If we did not know that $t_3$ cannot occur, then the largest empty trap in $C'$ is the empty set.*

There is a simple greedy algorithm to compute the largest empty siphon in a configuration [DE95, Example 4.5]): It starts with all empty states and then iteratively removes states that violate the siphon constraint. Similarly, to instead compute the largest non-empty trap, start with all non-empty states and then iteratively remove states that violate the trap constraint. By symbolically executing this algorithm, our overapproximation of reachability enforces that any initially empty $U$-siphon stays empty and any eventually empty $U$-trap starts empty.

### 4.2.2 Updating Stage Representation

When we find a new transition that is dead, we need to exclude all configurations from the stage where that transition is not dead. This is achieved by updating the basis of the upward-closed set $\uparrow\mathscr{B}$ such that $[\![\neg \mathit{Dead}(U)]\!] \subseteq \uparrow\mathscr{B}$ using Algorithm 1. It is based on the backward reachability algorithm that is commonly used for Petri nets (e.g., see [ACJ+96; FS01]). Intuitively, the algorithm first excludes configurations that enable dead transitions and then iteratively excludes configurations that can enable dead transitions via increasingly long transition sequences. We will now explain Algorithm 1 using an example.

**Algorithm 1** Updating Stage Representation via Backward Reachability

```
Require: population protocol $\mathcal{P} = (Q, T, I, O)$, basis $\mathscr{B}_0$ of upward-closed set, dead
    transitions $D \subseteq T$
Ensure: $\mathscr{B}$ is basis of upward closed set with $\uparrow\mathscr{B} \supseteq \uparrow\mathscr{B}_0$ and $[\![\neg \mathit{Dead}(U)]\!] \subseteq \uparrow\mathscr{B}$
 1: $\mathscr{B} := \mathscr{B}_0$
 2: $\mathscr{N} := \{{}^\bullet t \mid t \in D\}$                          ▷ Disable dead transitions
 3: while $\mathscr{N} \neq \emptyset$ do
 4:     $\mathscr{B} := \inf(\mathscr{B} \cup \mathscr{N})$          ▷ Keep minimal configurations of $\mathscr{B} \cup \mathscr{N}$
 5:     $\mathscr{N} := \emptyset$
 6:     for each $C \in \mathscr{B}$ and $t \in T \setminus D$ do
 7:         $C' := (C \ominus t^\bullet) + {}^\bullet t$                          ▷ Apply reverse of $t$
 8:         if $C' \notin \uparrow\mathscr{B}$ then
 9:             $\mathscr{N} := \mathscr{N} \cup \{C'\}$                        ▷ Exclude $\uparrow\{C'\}$
```

**Example 14** (Backward Reachability Algorithm)**.** *Recall the transitions of the majority voting protocol:*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad t_4 : y, n \mapsto y, y$$

*Let $\mathcal{S}$ be a stage with dead transitions $D = \{t_1\}$ and basis $\mathscr{B} = \{\wr Y, N \wr\}$. We will now use Algorithm 1 to compute the basis of a successor of $\mathcal{S}$ where the transitions $t_2$ and $t_4$ are dead as well. In a first step, we exclude configurations where some dead transition is enabled. The configurations that enable transition $t$ are exactly the configurations in $\uparrow\{{}^\bullet t\}$. Thus, we add the configurations $\wr Y, n \wr$ and $\wr y, n \wr$ to the basis (see Line 2). Whenever we add configurations to the basis, we keep only the minimal configurations (see Line 4). Because both new configurations are minimal, our new basis is $\{\wr Y, N \wr, \wr Y, n \wr, \wr y, n \wr\}$.*

*Next, the algorithm makes sure that the resulting stage is inductive. Thus, it also excludes configurations that can lead to already excluded configurations via a non-dead transition. The smallest configuration leading to a configuration $C'$ larger than some minimal configuration $C$ via a transition $t$ is $(C \ominus t^\bullet) + {}^\bullet t$. Thus, the algorithm adds $(C \ominus t^\bullet) + {}^\bullet t$ for each minimal configuration $C \in \mathscr{B}$ and non-dead transition $t \in T \setminus D$ to the basis (see Line 9).*[4] *This is repeated until the basis does not change anymore, i.e., until the stage is inductive. In our example,*

[4]Note that we apply the effect of the reversed transition $(C \ominus t^\bullet + {}^\bullet t)$ and not the usual effect of the transition $(C \ominus {}^\bullet t + t^\bullet)$. This gives rise to the name backward reachability algorithm.

*the only non-dead transition is $t_3$. Applying the reverse effect of $t_3$ to the new minimal configurations yields $\lbag Y, n \rbag \ominus t_3^\bullet + {}^\bullet t_3 = \lbag Y, N, y \rbag$ and $\lbag y, n \rbag \ominus t_3^\bullet + {}^\bullet t_3 = \lbag N, y, y \rbag$, of which only $\lbag N, y, y \rbag$ is not already excluded. The new basis is $\{\lbag Y, N \rbag, \lbag Y, n \rbag, \lbag y, n \rbag, \lbag N, y, y \rbag\}$. Because applying the reverse of $t_3$ to the new basis configuration results in $\lbag N, y, y, y \rbag$, which is already excluded by $\lbag N, y, y \rbag$, the algorithm terminates.*

### 4.2.3 Checking Stable Termination

To check that a stage $\mathcal{S}$ is terminal, we need to show that $\mathcal{S} \subseteq [\![\varphi^i_{\text{post}}]\!]$ for some postcondition $\varphi^i_{\text{post}}$ (see Property 4 of Definition 4). Thus, it is enough to show that the following Presburger formula is unsatisfiable:

$$\exists C \in \mathcal{S} : \neg \varphi^i_{\text{post}}(C)$$

### 4.2.4 Already Dead Transitions

Because stages are inductive, a transition $t$ is dead in every configuration of a stage $\mathcal{S}$ if and only if it is disabled in every configuration of $\mathcal{S}$. Thus, it is enough to show that the following Presburger formula is unsatisfiable:

$$\exists C \in \mathcal{S} : C \geq {}^\bullet t$$

### 4.2.5 Eventually Dead Transitions

Let $\mathcal{S}$ be the stage represented by $(D, \mathscr{B})$. Our goal is to find a certificate proving that a set of not-yet-dead transitions $U \subseteq T \setminus D$ eventually becomes dead in any fair execution starting in $S$. Then we know $\mathcal{S} \rightsquigarrow [\![Dead(U)]\!]$ and, because $\mathcal{S}$ is inductive, $\mathcal{S} \rightsquigarrow \mathcal{S} \cap [\![Dead(U \cup D)]\!]$. As deciding if there is a non-empty set of eventually dead transitions is PSPACE-hard [JLE77], we limit our search for certificates to *linear* functions $f(C) = a \cdot C = \sum_{q \in Q} a(q) \cdot C(q)$ that assign the *potential* $a(q) \in \mathbb{N}$ to each state $q$. We say that configuration $C$ has *potential* $f(C)$. Specifically, we search for two types of linear functions: *ranking functions* and *layer functions*.

**Ranking Functions.** A *ranking function* for transitions $U$ is a linear function such that (a) the occurrence of transitions in $U$ reduces the potential, and (b) the occurrence of other transitions does not increase the potential. This can be expressed as the following Presburger formulas:

$$\bigwedge_{u \in U} a \cdot \Delta(u) < 0 \tag{a}$$

$$\bigwedge_{t \in \overline{D \cup U}} a \cdot \Delta(t) \leq 0 \tag{b}$$

Note that a ranking function is a certificate for $\mathcal{S} \rightsquigarrow [\![Dead(U)]\!]$: Every configuration $C \in \mathcal{S}$ where $U$ is not dead can reach some configuration $C'$ via a transition sequence that contains a transition $t \in U$. As no transition increases the potential, but $t$ reduces the potential, it must hold that $f(C) > f(C')$.

**Layer Functions.** A *layer function* for transitions $U$ is a linear function such that (a) the occurrence of transitions in $U$ reduces the potential, and (b) when all transitions in $U$ become disabled, then they are dead. This can be expressed as the following Presburger formulas:

$$\bigwedge_{u \in U} a \cdot \Delta(u) < 0 \tag{a}$$

$$\bigwedge_{t \in \overline{D}} \bigwedge_{u \in U} \bigvee_{u' \in U} ({}^\bullet u \ominus t^\bullet) + {}^\bullet t \geq {}^\bullet u' \tag{b}$$

Intuitively, formula (b) says that if transition $t$ enabled the eventually dead transition $u$, then some eventually dead transition $u'$ was already enabled, i.e., no transition can reenable transitions in $U$ once they are all disabled. Note that a layer function is a certificate for $\mathcal{S} \rightsquigarrow [\![ Dead(U) ]\!]$: If the transitions in $U$ are not dead, then by (b) there is an enabled transition $t \in U$ and by (a) $t$ reduces the potential.

**Example 15** (Linear Certificates for Majority)**.** *Recall the majority protocol of Example 1 with its four transitions:*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad t_4 : y, n \mapsto y, y$$

*The stage graphs in Figure 4.1 prove its correctness. They use ranking and layer functions:*

- *The certificate $Y + N$ for $\mathcal{S}_1 \rightsquigarrow \mathcal{S}_2$ is a ranking function for $U = \{t_1\}$: No transition increases the number of agents in $\{Y, N\}$, but transition $t_1$ reduces the number.*
- *In $\mathcal{S}_2$, we know that $t_1$ and $t_2$ are already dead. The certificate $y$ for $\mathcal{S}_2 \rightsquigarrow \mathcal{S}_3$ is a layer function for $U = \{t_3\}$: Transition $t_3$ reduces the number of agents in $y$ and once there are no agents in $y$ the transition $t_4$ is also disabled. I.e., once $t_3$ is disabled, it is dead.*

### 4.2.6 Splitting

If we cannot find any eventually dead transitions for a stage $\mathcal{S}$ with representation $(D, \mathscr{B})$, we try to split it into smaller parts, i.e., we perform a case distinction. Specifically, we search for stages $\mathcal{S}_1 \ldots \mathcal{S}_n$ such that $\mathcal{S} = (\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_n)$ and for every $\mathcal{S}_i$ some additional transition $t_i \in T \setminus D$ is dead. Searching for a complete split in a single step entails solving a Presburger constraint that does not belong to the existential fragment. Therefore, we instead add one successor at a time until all configurations in stage $\mathcal{S}$ are covered.

A set of states $R \subseteq Q$ is a *siphon* for stage $\mathcal{S}$ if for every transition $t \in T \setminus D$ it holds that $t^\bullet \cap R \neq \emptyset \implies {}^\bullet t \cap R \neq \emptyset$. In other words, every non-dead transition that produces an agent in siphon $R$ consumes an agent in $R$. Intuitively, once $R$ is empty, it stays empty, i.e., an empty siphon for $\mathcal{S}$ is a *death certificate* for all transitions $t$ with $^\bullet t \cap R \neq \emptyset$.

Every successor we add corresponds to a siphon $R \subseteq Q$ for stage $\mathcal{S}$ that certifies the death of some non-dead transition and is empty in some not yet covered configuration $C$. Specifically, the new successor contains all configurations of $\mathcal{S}$ where the siphon $R$ is empty. The set of configurations where the siphon $R$ is *not* empty is upward-closed with basis $\{\wr q \wr \mid q \in R\}$. Thus, we can limit the current stage to configurations where $R$ is empty by updating the basis $\mathscr{B}$ as described in Section 4.2.2. To ensure that we construct splits with few successors, we use a heuristic for choosing the siphon of the next successor. We will explain and motivate our heuristic with the following example.

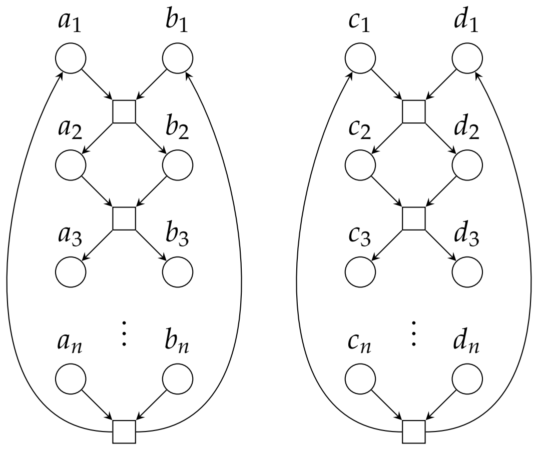


**Figure 4.10:** Petri net visualization of population protocol used to explain the heuristic for the splitting of stages. This system has a large number of siphons. For example, any set of states $R \subseteq \{a_1, \ldots, a_n, b_1, \ldots, b_n\}$ that contains at least one state of $\{a_i, b_i\}$ for each level $i$ is a siphon. If $R$ is empty, it stays empty and thus all transitions on the left are dead.

Consider the population protocol visualized in Figure 4.10 with states $A \cup B \cup C \cup D$ where $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, $C = \{c_1, \ldots, c_n\}$ and $D = \{d_1, \ldots, d_n\}$ and transitions

$$\begin{aligned} &\{a_i, b_i \mapsto a_{i+1}, b_{i+1} \mid 1 \leq i < n\} \cup \{a_n, b_n \mapsto a_1, b_1\} \\ \cup &\{c_i, d_i \mapsto c_{i+1}, d_{i+1} \mid 1 \leq i < n\} \cup \{c_n, d_n \mapsto c_1, d_1\}. \end{aligned}$$

Assume that the current stage contains exactly those configurations where either all states in $A \cup B$ are empty or all states in $C \cup D$ are empty. Note that no transition is dead in every configuration, i.e., we need to split the stage. The split with the fewest successors has size two and uses the empty siphons

$A \cup B$ (e.g., in configuration $\wr c_1, \ldots c_n, d_1, \ldots, d_n \wr$) and
$C \cup D$ (e.g., in configuration $\wr a_1, \ldots a_n, b_1, \ldots, b_n \wr$).

However, there are many siphons that are empty in some configuration of the stage and if we do not choose siphons carefully, the resulting split can have an exponential number of successors. For example, we could choose:

$$\begin{array}{rll}
1. & \{a_1, a_2, a_3, \dots, a_n\} & \text{in } \Lbag b_1, b_2, b_3, \dots, b_n \Rbag \\
2. & \{b_1, a_2, a_3, \dots, a_n\} & \text{in } \Lbag a_1, b_2, b_3, \dots, b_n \Rbag \\
3. & \{a_1, b_2, a_3, \dots, a_n\} & \text{in } \Lbag b_1, a_2, b_3, \dots, b_n \Rbag \\
\vdots & \vdots & \vdots \\
2^n. & \{b_1, b_2, b_3, \dots, b_n\} & \text{in } \Lbag a_1, a_2, a_3, \dots, a_n \Rbag \\
2^n + 1. & C \cup D & \text{in } \Lbag a_1, a_2, a_3, \dots, a_n, b_1, b_2, b_3, \dots, b_n \Rbag
\end{array}$$

Note that the split has size $2^n + 1$ and was the result of always choosing a siphon with the fewest states.[5] This split missed some important correlation: For example, when the siphon $A$ is empty in a configuration of the stage, then so was either the larger siphon $A \cup B$ or the larger siphon $A \cup C \cup D$. This is why we *consider only siphons that are the largest empty siphon in some configuration*. Still, if we would choose any such siphon, we could choose:

$$\begin{array}{rll}
1. & \{a_1, a_2, a_3, \dots, a_n\} \cup C \cup D & \text{in } \Lbag b_1, b_2, b_3, \dots, b_n \Rbag \\
2. & \{b_1, a_2, a_3, \dots, a_n\} \cup C \cup D & \text{in } \Lbag a_1, b_2, b_3, \dots, b_n \Rbag \\
3. & \{a_1, b_2, a_3, \dots, a_n\} \cup C \cup D & \text{in } \Lbag b_1, a_2, b_3, \dots, b_n \Rbag \\
\vdots & \vdots & \vdots \\
2^n. & \{b_1, b_2, b_3, \dots, b_n\} \cup C \cup D & \text{in } \Lbag a_1, a_2, a_3, \dots, a_n \Rbag \\
2^n + 1. & C \cup D & \text{in } \Lbag a_1, a_2, a_3, \dots, a_n, b_1, b_2, b_3, \dots, b_n \Rbag
\end{array}$$

Note that the split has size $2^n + 1$ and was the result of always choosing a siphon with the most states. While we do not miss any correlation this way, we cover only few configurations with each additional successor, as larger siphons are empty in fewer configurations. This is why we always *choose a smallest siphon that is the largest empty siphon in a configuration* of the stage. Note that this results exactly in the minimal split of size two.

### 4.2.7 Automatic Speed Bounds

So far, we only use stage graphs to automatically verify that a population protocol satisfies a stable termination property. However, in practice, we are often also interested in the time it takes for the protocol to stabilize under stochastic scheduling (see Section 3.3). Instead of just verifying that a population protocol computes a Presburger formula, we also want to analyze the speed of the computation. In this section, we argue that the certificates of a stage graph contain information about the speed of a population protocol. Specifically, they can prove that the expected number of interactions

[5]While the empty set is the smallest empty siphon, it is no death certificate for any transitions.

needed to enter a successor is bounded by a function that depends only on the number of agents $n$. Note that the following approach is based on [BEK18] where Blondin *et al.* use a closely related version of stage graphs to automatically bound the speed of population protocols.

We produce four different speed bounds: $2^{\mathcal{O}(n \log n)}$, $\mathcal{O}(n^c)$ for some constant $c \geq 3$, $\mathcal{O}(n^3)$, and $\mathcal{O}(n^2 \log n)$. Note that all speed bounds are upper bounds for the expected number of transitions, e.g., a certificate might guarantee only a speed of $\mathcal{O}(n^3)$ even though the actual speed of the stage is $n^2$. Each of the four speed bounds corresponds to a different class of certificates. As before, certificates are linear functions $f(C) = a \cdot C = \sum_{q \in Q} a(q) \cdot C(q)$ that assign the *potential* $a(q) \in \mathbb{N}$ to each state $q$. We say that configuration $C$ has the *potential* $f(C)$, a state $q$ *has potential* if $a(q) > 0$, and an agent *has potential* if it is in a state with potential.

**Layer Functions - $2^{\mathcal{O}(n \log n)}$.**
Layer functions are one of the two types of certificates already described in Section 4.2.5. Recall that a *layer function* for transitions $U$ is a linear function such that (a) the occurrence of transitions in $U$ reduces the potential, and (b) when all transitions in $U$ become disabled, then they are dead. We will now argue that a layer function guarantees the speed $2^{\mathcal{O}(n \log n)}$: As long as the transitions in $U$ are not disabled, it is possible to reduce the potential due to (a). Intuitively, a layer function is a bounded certificate with bound 1. A reduction of the potential can happen only $\mathcal{O}(n)$ times in a row because the maximum potential is linear in $n$ while the reduction is at least constant. Thus, for any configuration, there is a transition sequence of length $\mathcal{O}(n)$ that disables $U$, and at that point, (b) tells us that the transitions in $U$ are dead. The probability of executing a specific transition is $1/n^2$ and the probability of executing a sequence of length $\mathcal{O}(n)$ is $n^{-2\mathcal{O}(n)} = 2^{-\mathcal{O}(n \log n)}$. Thus, we expect $2^{\mathcal{O}(n \log n)}$ transitions until the successor is entered.

**Ranking Functions - $\mathcal{O}(n^c)$.**
Ranking functions are one of the two types of certificates already described in Section 4.2.5. A *ranking function* for transitions $U$ is a linear function such that (a) the occurrence of transitions in $U$ reduces the potential, and (b) the occurrence of other transitions does not increase the potential. We will now argue that a ranking function guarantees the speed $\mathcal{O}(n^c)$ for some constant $c \geq 3$: As long as the transitions in $U$ are not disabled, it is possible to reduce the potential due to (a). This can only happen $\mathcal{O}(n)$ times because the potential is at most linear in $n$, never increases, and is always reduced by at least some constant value. Because the configurations that can enable a transition $t$ are upward-closed, it is always possible to enable a transition in $U$ after a constant number $d \in \mathbb{N}$ of transitions.[6] Thus, as long as the transitions in $U$ are not dead, it is possible to execute some transition $t \in U$ in $d+1$ steps by enabling $t$ in $d$ steps and then executing $t$. Intuitively, a ranking function is a bounded certificate with

[6]The constant $d$ can be computed via the backward reachability algorithm. Crucially, while $d$ can be double-exponential in the size of the protocol, it is independent of the input size $n$.

bound $d+1$. The probability of executing a sequence of $d+1$ steps is $n^{-2(d+1)}$. Therefore, we expect the $\mathcal{O}(n)$ reductions to occur within $\mathcal{O}(n) \cdot n^{2d+2} = \mathcal{O}(n^c)$ transitions where $c = 2d+3$.

**Layer Ranking Functions - $\mathcal{O}(n^3)$.**
Intuitively, a layer ranking function is both a ranking function and a layer function. Formally, a *layer ranking function* for transitions $U$ is a ranking function for $U$ such that (c) when all transitions in $U$ become disabled, then they are dead. Due to (c), either the transitions in $U$ are dead and we are in a successor stage, or some transition in $U$ is already enabled. Thus, we have $d = 0$ and the speed is $\mathcal{O}(n^c) = \mathcal{O}(n^{2d+3}) = \mathcal{O}(n^3)$.

**Fast Layer Ranking Functions - $\mathcal{O}(n^2 \log n)$.**
A layer ranking function is *fast* if every agent that has potential can always interact with another agent in a way that reduces the potential. Intuitively, this implies that as long as the potential is large, the probability of reducing the potential is large. Because the potential is at most linear in $n$ and is distributed among $n$ agents, the probability of choosing an agent with potential is approximately $\frac{f(C)}{\mathcal{O}(n)}$. As this agent can reduce the potential with some other agent that is chosen with probability $1/n$, the probability of reducing the potential is approximately $f(C)/n^2$. The potential $f(C)$ can be reduced at most $\mathcal{O}(n)$ times because the potential is always reduced by at least some constant value. Thus, the expected number of transitions is $\sum_{i=1}^{\mathcal{O}(n)} \frac{n^2}{i} = n^2 \sum_{i=1}^{\mathcal{O}(n)} \frac{1}{i} = \mathcal{O}(n^2 \log n)$.

**Algorithm 2** Finding Eventually Dead Transitions and Speed Bounds

```
1:  Search ranking function $f_r$ for largest set $U_r$ of transitions
2:  if $U_r = \emptyset$ then
3:      Search layer function $f_l$ for largest set $U_l$ of transitions
4:      if $U_l = \emptyset$ then
5:          return $\emptyset$                                  ▷ no certificate, no speed bound
6:      else
7:          return $U_l$ with certificate $f_l$ and speed $2^{\mathcal{O}(n \log n)}$
8:  else
9:      Construct layer ranking function $f_{lr}$ from $f_r$ for largest set $U_{lr}$ of transitions
10:     if $U_{lr} = \emptyset$ then
11:         return $U_r$ with certificate $f_r$ and speed $\mathcal{O}(n^c)$
12:     else
13:         Search for fast layer ranking function $f_{flr}$ for $U_{lr}$
14:         if no fast layer ranking function was found then
15:             return $U_{lr}$ with certificate $f_{lr}$ and speed $\mathcal{O}(n^3)$
16:         else
17:             return $U_{lr}$ with certificate $f_{flr}$ and speed $\mathcal{O}(n^2 \log n)$
```

**Algorithm.**
Algorithm 2 shows our heuristic for searching eventually dead transitions with certificates that have fast speeds. The construction of the largest ranking function in Line 1 first searches ranking functions for each transition individually and then combines them. The construction of the largest layer function in Line 3 searches layer functions for increasingly large sets of transitions. In Line 9, we construct the largest layer ranking function from the largest ranking function by iteratively removing transitions that violate the layer constraint (b). The search for fast layer ranking functions in Line 13 repeatedly checks if the current function is fast. If it is not fast due to agents in some state $q$, it looks for an alternative function where $q$ does not have potential.

**Experimental Results.**
We implemented our algorithm on top of the SMT solver Z3 [MB08] and evaluate the automatic speed analysis of population protocols in Table 4.1. To allow an easy comparison with [BEK18], we use the same benchmarks and restate their evaluation results. The experiment was carried out on a machine with an Intel Core i7-11700K @ 3.60GHz CPU, 8GB of RAM, and a timeout of 1000 seconds ($\approx$ 16.67 minutes).

For most benchmarks, both approaches yield identical speed bounds. The only exception is the average-and-conquer protocol [AGV15] with parameters $m$=5 and $d$=1, where we can show a tighter speed bound of $\mathcal{O}(n^2 \log n)$ instead of $\mathcal{O}(n^3)$.[7] Our approach outperforms [BEK18] significantly across all benchmarks and exhibits superior scalability as the number of states and transitions increases. For instance, we can analyze the logarithmic flock-of-birds protocol [BEK18; BEJ18a] up to parameter $c$=$2^{45}-1$, while [BEK18] times out for $c$=$2^{12}-1$. One contributing factor is that the stage graphs generated by [BEK18] contain many stages. In the case of the flock-of-birds protocol [CDF+11], they already generate 54 stages for $c$=5, and this count grows rapidly, reaching 12294 stages for $c$=13. In contrast, our stage graphs usually consist of 3-8 stages, and this number remains relatively constant even when a protocol's parameters are altered. This small size of our stage graphs makes it possible to gain valuable insights about a protocol's computation, as described in next Section 4.3.

[7]The exact reason for this difference is unknown. It might be the case that the lower number of stages helps or that our search for fast layer ranking functions is simply stronger. In general, the average-and-conquer protocol [AGV15] is a very interesting benchmark for automatic speed analysis as there seems to be a non-trivial border between cases with $\mathcal{O}(n^2 \log n)$ and $\mathcal{O}(n^3)$ bounds (compare Table 4.1).

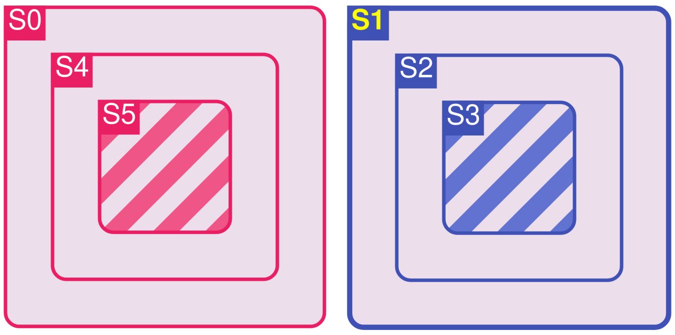
**Table 4.1:** Evaluation comparing the speed analysis of our approach and [BEK18] with timeout 1000s. $|Q|$, $|T|$, and $|\mathcal{S}|$ are the number of states, transitions, and stages.

| Protocol | | | **[BEK18]** | | | **This work** | | |
|---|---|---|---|---|---|---|---|---|
| Parameters | $\|Q\|$ | $\|T\|$ | $\|\mathcal{S}\|$ | Speed | Time | $\|\mathcal{S}\|$ | Speed | Time |
| Broadcast [CDF+11] | 2 | 1 | 5 | $\mathcal{O}(n^2 \log n)$ | $< 1$s | 3 | $\mathcal{O}(n^2 \log n)$ | $< 1$s |
| Maj. [BEK18, Example 3] | 5 | 6 | 13 | $\mathcal{O}(n^2 \log n)$ | $< 1$s | 6 | $\mathcal{O}(n^2 \log n)$ | $< 1$s |
| Majority (Example 5)[a] | 4 | 3 | 9 | $\mathcal{O}(n^2 \log n)$ | $< 1$s | 6 | $\mathcal{O}(n^2 \log n)$ | $< 1$s |
| Majority (Example 1) | 4 | 4 | 11 | $2^{\mathcal{O}(n \log n)}$ | $< 1$s | 6 | $2^{\mathcal{O}(n \log n)}$ | $< 1$s |
| Flock-of-bird protocol [AAD+06] (see Example 7)[b]: $x \geq c$ | | | | | | | | |
| $c = 30$ | 31 | 496 | 126 | $\mathcal{O}(n^3)$ | 119$s$ | 3 | $\mathcal{O}(n^3)$ | 17$s$ |
| $c = 50$ | 51 | 1326 | 206 | $\mathcal{O}(n^3)$ | 953$s$ | 3 | $\mathcal{O}(n^3)$ | 140$s$ |
| $c = 70$ | 71 | 2556 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 701$s$ |
| Logarithmic flock-of-bird protocol [BEK18] adapted from [BEJ18a][b]: $x \geq c$ | | | | | | | | |
| $c = 2^5 - 1$ | 10 | 34 | 130 | $\mathcal{O}(n^3)$ | 6$s$ | 3 | $\mathcal{O}(n^3)$ | $< 1$s |
| $c = 2^{10} - 1$ | 20 | 119 | 4098 | $\mathcal{O}(n^3)$ | 396$s$ | 3 | $\mathcal{O}(n^3)$ | 3$s$ |
| $c = 2^{25} - 1$ | 50 | 674 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 52$s$ |
| $c = 2^{40} - 1$ | 80 | 1679 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 408$s$ |
| $c = 2^{45} - 1$ | 90 | 2114 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 702 |
| Flock-of-bird protocol (tower variant) [CDF+11][b]: $x \geq c$ | | | | | | | | |
| $c = 10$ | 11 | 19 | 1542 | $\mathcal{O}(n^3)$ | 84$s$ | 3 | $\mathcal{O}(n^3)$ | $< 1$s |
| $c = 13$ | 14 | 25 | 12294 | $\mathcal{O}(n^3)$ | 816$s$ | 3 | $\mathcal{O}(n^3)$ | $< 1$s |
| $c = 15$ | 16 | 29 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | $< 1$s |
| $c = 20$ | 21 | 39 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 18$s$ |
| $c = 23$ | 24 | 45 | - | - | T/O | 3 | $\mathcal{O}(n^3)$ | 409$s$ |
| Average-and-conquer protocol [AGV15][a]: $x \geq y$ (Majority) | | | | | | | | |
| $m = 3, d = 2$ | 8 | 36 | 1948 | $\mathcal{O}(n^2 \log n)$ | 99$s$ | 8 | $\mathcal{O}(n^2 \log n)$ | 2$s$ |
| $m = 5, d = 1$ | 8 | 36 | 1870 | $\mathcal{O}(n^3)$ | 80$s$ | 6 | $\mathcal{O}(n^2 \log n)$ | 2$s$ |
| $m = 5, d = 2$ | 10 | 55 | - | - | T/O | 8 | $\mathcal{O}(n^3)$ | 4$s$ |
| $m = 7, d = 1$ | 10 | 55 | - | - | T/O | 6 | $\mathcal{O}(n^2 \log n)$ | 3$s$ |
| $m = 7, d = 2$ | 12 | 78 | - | - | T/O | 8 | $\mathcal{O}(n^3)$ | 5$s$ |
| $m = 9, d = 1$ | 12 | 78 | - | - | T/O | 6 | $\mathcal{O}(n^3)$ | 6$s$ |
| $m = 39, d = 1$ | 42 | 903 | - | - | T/O | 6 | $\mathcal{O}(n^3)$ | 515$s$ |
| $m = 15, d = 14$ | 44 | 990 | - | - | T/O | 8 | $\mathcal{O}(n^3)$ | 942$s$ |
| Remainder protocol [AAD+06]: $\sum_{1 \leq i < m} a_i \cdot x_i \equiv 0 \pmod{m}$ | | | | | | | | |
| $m = 5$ | 7 | 25 | 225 | $\mathcal{O}(n^2 \log n)$ | 13$s$ | 6 | $\mathcal{O}(n^2 \log n)$ | 2$s$ |
| $m = 9$ | 11 | 63 | 7035 | $\mathcal{O}(n^2 \log n)$ | 544$s$ | 6 | $\mathcal{O}(n^2 \log n)$ | 5$s$ |
| $m = 10$ | 12 | 75 | - | - | T/O | 6 | $\mathcal{O}(n^2 \log n)$ | 7$s$ |
| $m = 20$ | 27 | 150 | - | - | T/O | 6 | $\mathcal{O}(n^2 \log n)$ | 40$s$ |
| $m = 30$ | 22 | 250 | - | - | T/O | 6 | $\mathcal{O}(n^2 \log n)$ | 901$s$ |
| Threshold protocol [AAD+06]: $\sum_{-2 \leq i \leq 2} a_i \cdot x_i < c$ | | | | | | | | |
| $c = 0$ | 20 | 146 | 1049 | $\mathcal{O}(n^3)$ | 166$s$ | 6 | $\mathcal{O}(n^3)$ | 7$s$ |
| $c = 1$ | 28 | 288 | 1049 | $\mathcal{O}(n^3)$ | 155$s$ | 6 | $\mathcal{O}(n^3)$ | 14$s$ |
| $c = 2$ | 36 | 478 | - | - | T/O | 6 | $\mathcal{O}(n^3)$ | 25$s$ |
| $c = 4$ | 52 | 1002 | - | - | T/O | 6 | $\mathcal{O}(n^3)$ | 129$s$ |
| $c = 6$ | 68 | 1718 | - | - | T/O | 6 | $\mathcal{O}(n^3)$ | 727$s$ |

[a]Under the assumption that there is no tie.
[b]We use $[\![Dis(U)]\!]$ as overapproximation of $[\![Dead(U)]\!]$ as described in Sec. 5 of [BEH+20] (see App. A).

## 4.3 Tool: Peregrine

We implemented the automatic verification procedure of Section 4.2 in the free tool Peregrine.[8] In Peregrine, users can create, simulate, analyze, and verify population protocols with an easy-to-use graphical user interface. We will now highlight a few of the features of Peregrine that are related to stage graphs. These features help users to understand how a protocol works, how fast it is, and why it is correct or incorrect.[9]

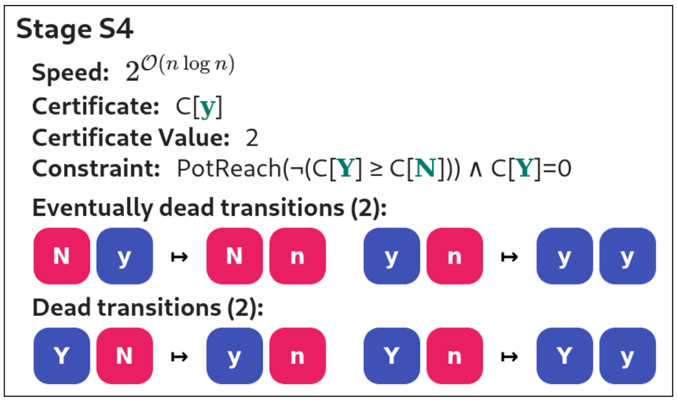


| Stage | Constraint | Dead | Cert. | Speed |
|---|---|---|---|---|
| $\mathcal{S}_0$ | $\mathit{PotReach}(\llbracket Y < N \rrbracket)$ | $\emptyset$ | $Y$ | $\mathcal{O}(n^2 \log n)$ |
| $\mathcal{S}_4$ | $\mathit{PotReach}(\llbracket Y < N \rrbracket) \land Y = 0$ | $\{t_1, t_2\}$ | $y$ | $2^{\mathcal{O}(n \log n)}$ |
| $\mathcal{S}_5$ | $\mathit{PotReach}(\llbracket Y < N \rrbracket) \land Y = 0 \land y = 0$ | $\{t_1, t_2, t_3, t_4\}$ | $\perp$ | $\perp$ |
| $\mathcal{S}_1$ | $\mathit{PotReach}(\llbracket Y \geq N \rrbracket)$ | $\emptyset$ | $N$ | $\mathcal{O}(n^2 \log n)$ |
| $\mathcal{S}_2$ | $\mathit{PotReach}(\llbracket Y \geq N \rrbracket) \land N = 0$ | $\{t_1, t_3\}$ | $n$ | $\mathcal{O}(n^2 \log n)$ |
| $\mathcal{S}_3$ | $\mathit{PotReach}(\llbracket Y \geq N \rrbracket) \land N = 0 \land n = 0$ | $\{t_1, t_2, t_3, t_4\}$ | $\perp$ | $\perp$ |

**Figure 4.11:** Stage graphs visualization of Peregrine for majority voting protocol of Example 1. For each stage, the tool gives a constraint, a set of dead transitions, a certificate and a speed. The combined speed of the protocol is $2^{\mathcal{O}(n \log n)}$ because of the slow stage $\mathcal{S}_4$.

**Stage Graph Visualization.** For the majority voting protocol (see Example 1), the tool verifies correctness in less than two seconds by generating two stage graphs. It then visualizes the stage graphs as Venn diagrams (see Figure 4.11) to highlight that the system becomes trapped within sets of configurations with decreasing size. The user clearly sees that the protocol works in three phases. The configurations of each stage are described with an easy-to-read Presburger constraint. For each stage, Peregrine lists the already dead transitions. If a stage is non-terminal, Peregrine additionally shows the eventually dead transitions with the corresponding certificate and implied speed. This lets the user know that the protocol is fast in most phases of the computation but

[8]Peregrine is available at https://peregrine.model.in.tum.de together with an online demo.
[9]Peregrine is developed by multiple authors. As detailed in Appendix B, not all of the following features were implemented by the author of this thesis.



**Stage S4**

**Speed:** $2^{\mathcal{O}(n \log n)}$
**Certificate:** C[**y**]
**Certificate Value:** 2
**Constraint:** PotReach(¬(C[**Y**] ≥ C[**N**])) ∧ C[**Y**]=0

**Eventually dead transitions (2):**

N y ↦ N n    y n ↦ y y

**Dead transitions (2):**

Y N ↦ y n    Y n ↦ Y y

**Figure 4.12:** Details given py Peregrine for stage $\mathcal{S}_4$ in Figure 4.11 at configuration $\wr N, 4 \cdot n, 2 \cdot y \wr$.

slow in $\mathcal{S}_4$, which has the speed $2^{\mathcal{O}(n \log n)}$. The details of $\mathcal{S}_4$ (see Figure 4.12) explain its slow speed: The two eventually dead transitions $t_3 : N, y \mapsto N, n$ and $t_4 : y, n \mapsto y, y$ have opposite effects.

**Simulation within Stage Graphs.** To understand the computation of a population protocol even better, Peregrine allows the user to simulate the protocol while visualizing the run in the Venn diagram of the stage graphs (see Figure 4.13). First, the user chooses an initial configuration or starts the simulation from the precomputed example configuration of a stage. The current configuration is always shown as a yellow circle within the region of its stage. Then, the user chooses the next interaction by clicking on two agents in the current configuration. The resulting configuration is automatically placed in the correct stage and connected with an arrow. Step by step this builds up a partial reachability graph. For configurations in non-terminal stages, Peregrine also

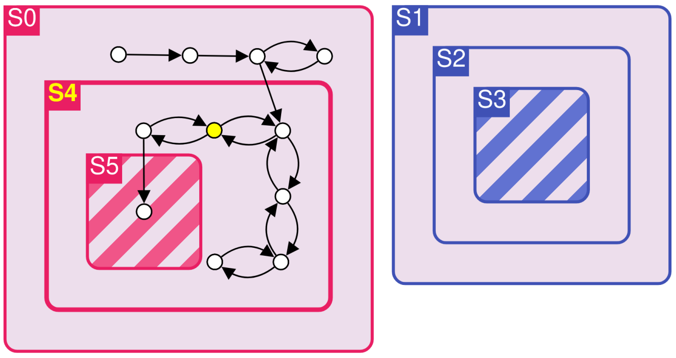


**Figure 4.13:** Peregrine's visualization of a simulation for the majority voting protocol inside of the protocol's stage graphs. The simulation is a walk in the partially constructed reachability graph. Configurations of a stage are automatically placed within the stage's region in the Venn diagram. The currently selected configuration is ○ = $\wr N, 4 \cdot n, 2 \cdot y \wr$.

shows the current value of the certificate. Intuitively, this value describes the distance to the next stage and transitions that reduce this value effectively make progress. The user can automatically apply such a transition by clicking the PROGRESS button. To generate a full simulation of the protocol with random interactions, the user can click on PLAY. It is possible to navigate through the current simulation run by clicking on the buttons PREV and NEXT or by selecting a configuration in the Venn diagram.

**Incorrect Protocols.** In case a user tries to verify an incorrect population protocol, Peregrine will fail to find stage graphs that verify its correctness. Instead, it will show partially constructed stage graphs and highlight the problematic stages. For example, if we remove the transition $t_4 : y, n \mapsto y, n$ from the majority voting protocol, then the protocol no longer computes majority. The partial stage graph for this broken majority protocol is depicted in Figure 4.14. The user can easily see that the protocol is still correct in case there was a majority for "no" because the left stage graph is complete. Peregrine failed to prove correctness only for a single stage. The constraint for the problematic stage $\mathcal{S}_4$ is $PotReach(\llbracket Y \geq N \rrbracket) \land Y = 0 \land N = 0$, i.e., the protocol might be incorrect for configurations with no active agents. Using this information, Peregrine was able to find a counterexample for correctness by querying the tool LoLA [Wol18]. The counterexample starts in state $\lbag Y, N \rbag$ and ends in $\lbag y, n \rbag$ without consensus. This makes it easy to realize that the protocol is incorrect in case of a tie.

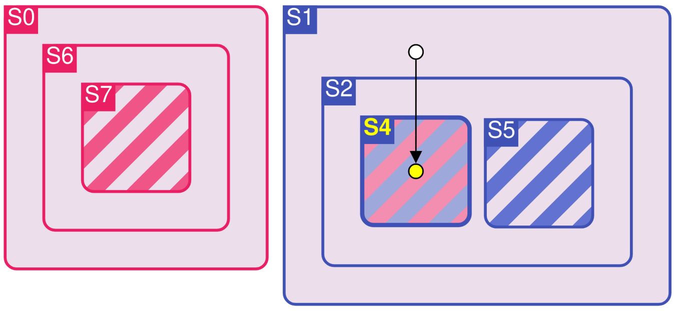


**Figure 4.14:** Counterexample automatically found by Peregrine when verifying an incorrect majority voting protocol. It is shown in the stage graphs as a run from ○ = $\lbag Y, N \rbag$ to ● = $\lbag y, n \rbag$. The graph with root $\mathcal{S}_1$ is only a partial stage graph, because stage $\mathcal{S}_4$ contains configurations that do not have the correct consensus. Stage S3 is not visualized as it was split and thus is completely covered by S4 and S5.

## 4.4 Related Work

Early work on verification of population protocols can be divided into two main categories. The first category focuses on verification for a finite number of inputs, which involves techniques such as model checking and the construction of reachability graphs for fixed inputs [PLD08; SLD+09; CMS10; CDF+11]. While this approach can demonstrate partial correctness or identify counterexamples, it does not offer complete verification for all possible inputs. The second category involves the use of an interactive theorem prover to verify protocols, as demonstrated in [DM09]. However, this approach is semi-automatic and requires human interaction for each new protocol.

The first efficient algorithm for the automatic verification of population protocols was described in [BEJ+17]. It combines the techniques of potential reachability (compare Section 4.2.1) and layered termination (compare layer functions in Section 4.2.5) to verify the correctness of a subclass of population protocols. Notably, this subclass contains only protocols that are silent, i.e., that always reach a configuration where all non-silent transitions are disabled. The approach was the first verification technique implemented in the tool Peregrine [BEJ18b]. Our approach can be understood as a generalization of [BEJ+17] in the sense that their technique corresponds to a linear stage graph where all certificates are layer functions. Next to the larger class of verifiable protocols, the most significant difference is the fact that we supply a certificate for correctness that can be independently checked.

A first version of stage graphs was presented in [BEK18] in order to automatically analyze the speed of population protocols. While their approach cannot be used to show correctness of population protocols, it can show termination, i.e., it can verify the property:

$$\mathbf{F}(\mathbf{G}\varphi_{\text{cons}}^{true} \vee \mathbf{G}\varphi_{\text{cons}}^{false})$$

They describe a procedure that constructs a stage graph using heuristics, such as a technique based on the transformation graph, which describes how transitions alter the states of agents. In comparison, our work gives the first completeness result that guarantees the existence of a stage graph for the larger class of stable termination properties. Consequently, our approach can verify the correctness of population protocols while providing the same speed analysis. Using better heuristics, we construct stage graphs more efficiently and produce significantly fewer and simpler stages (see Section 4.2.7). As a result, our stage graphs serve not only as certificates for correctness but also help to understand how a protocol computes and why it is correct.

### Other Parameterized Verification Techniques

There is a large body of research that verifies parametric systems using a cutoff technique. Relevant references include [ARZ+15; BCG89; CTT+04; EN03; KKW10], and a comprehensive survey can be found in [BJK+15]. The idea of this technique is to prove that a specification holds for any number of agents as long as it holds for a number of agents below a specific threshold, known as the cutoff. This approach is only applicable if there is a cut-off, and it is only effective if this cutoff is relatively small. While the cutoff technique can be used to verify parametric systems with an array or ring structure, they are not effective for population protocols.

Another model checking technique that effectively handles the infinite state space of parameterized systems is regular model checking [BJN+00; Abd12]. The fundamental concept behind this approach is to represent the system configuration as a string, where each agent is denoted by a letter. This representation enables the use of regular languages to describe infinite sets of configurations and regular transducers to model the transition relation. In combination with automata learning [Ang87], a technique that learns regular languages from examples, regular model checking was used to prove safety properties [CHL+17], liveness properties under arbitrary schedulers [LR16], and even termination under so-called finitary fairness [LLM+17]. However, population protocols possess both an incompatible communication structure (clique) and a different fairness assumption (global fairness).

### Population Protocol Simulation

Simulation serves as a valuable tool for gaining insights into stochastic population protocols, particularly in the analysis of their speed. Sequential simulation involves the random selection of the next pair of interacting agents for each interaction. One such sequential simulator is incorporated into the Peregrine tool, which is described in Section 4.3. Alternatively, a more efficient batch simulation technique, outlined in [BHK+20], allows for multiple interactions to occur simultaneously. This technique considers interaction sequences in which no agent participates more than once, resulting in accelerated simulation while maintaining the precise stochastic dynamics. The implementation of this approach can be found in the software package *ppsim*, which facilitates efficient simulation and visualization of population protocols [DS21].

## 4.5 Open Research Questions

Stage graphs verify stable termination properties for a single population protocol. However, in many cases, protocols are parameterized, resulting in a family of similar protocols. For instance, consider the flock-of-birds family in Example 7: Each parameter $c \geq 1$ defines a different protocol that computes the formula $q_1 \geq c$. Hence, an important area to explore is the verification of protocol families. Specifically, it is worth investigating if stage graphs can be extended to certificates for protocol families. This may result in an automatic and efficient technique that can be implemented in the tool Peregrine.

Many extensions of the population protocol model have been introduced. These include cover-time services [MS15b], clocks [Asp17], broadcasts [BEJ19], and protocols where the number of states depends on the number of agents [AAE+17; GS20]. Future work may investigate how stage graphs can be used to prove properties of these extended models.

Another interesting research direction concerns the tailor problem:

**Definition 9** (Tailor Problem)**.**
*Given: A population protocol $\mathcal{P}$ that computes some formula.*
*Question: What formula does $\mathcal{P}$ compute?*

Esparza *et al.* already showed that the tailor problem is decidable and they even give an algorithm that solves it [EGL+17]. However, this algorithm has very limited practical application because of its high computational complexity and large resulting formulas. Future research may find a more efficient and practical technique that can be used to provide helpful feedback during the design process of population protocols. For example, it may be possible to automatically generate a stage graph for consensus ($\mathbf{F}(\mathbf{G}\varphi_{\text{cons}}^{\mathit{true}} \vee \mathbf{G}\varphi_{\text{cons}}^{\mathit{false}})$) and then extract and simplify the computed formula.

The automatic computation of speed bounds using stage graphs provides an upper bound on the expected number of interactions until protocol convergence. Consequently, the speed analysis of population protocols could be improved by a complementary lower bound.

# 5 Synthesis of Population Protocols

The synthesis of population protocols involves the creation of a protocol that satisfies a formal specification. The specification is the Presburger formula that the protocol should compute. While the synthesis task differs from the verification task discussed in Chapter 4, both are closely related. Verification checks the correctness of a protocol, whereas synthesis aims to construct a correct protocol.

Although the focus of this work primarily lies in the efficient analysis of population protocols, it is worth mentioning that we have also explored the synthesis aspect. While synthesis is not the central topic of our research, it is closely tied to analysis, and we believe it is valuable to provide a high-level overview of our work in this area.

## Problem Statement

Recall that population protocols compute precisely the formulas expressible in Presburger arithmetic [AAE+07]. Because Presburger arithmetic admits quantifier elimination, the synthesis task is as follows:

**Definition 10** (Synthesis Task)**.**
*Input:* *A quantifier-free Presburger formula* $\varphi$*.*
*Output:* *A population protocol* $\mathcal{P}$ *that computes* $\varphi$*.*

For the rest of the chapter, let $\varphi$ be a quantifier-free Presburger formula that is the input for a synthesis procedure. We evaluate synthesis procedures using two metrics:

The first is the *size* of the resulting protocol, i.e., its number of states. This is compared to $|\varphi|$, which is the length of the formula $\varphi$ with constants written in binary. For example, the formula $42x - 3y > 7 \lor 8x - 1y \equiv_{11} 3$ with binary constants is written as the string "$101010x - 11y > 111 \lor 1000x - 1y \equiv_{1011} 11$" which has a length of 31. A protocol is considered *succinct* if its size is $\mathcal{O}(\text{poly}(|\varphi|))$, i.e., if the number of states is polynomial in the length of the formula.

The second metric is the *speed* of the protocols, i.e., the expected number of interactions required to reach a stable consensus. In the context of this chapter, we consider a protocol *fast* if its speed is $\mathcal{O}(n^2 \log n)$, where $n$ is the number of agents.[1]

[1] We will actually synthesize protocols with speed $\mathcal{O}(n^2)$. However, this weaker definition of "fast" makes it easier to compare with competing synthesis procedures and it coincides with the notion of fast layer ranking functions in Section 4.2.7.

## 5.1 The First Procedure: Fast but Large

The first synthesis procedure was given by Angluin *et al.* [AAD+04] as part of their proof about the expressive power of population protocols [AAE+07]. We will now give a brief description of this procedure and explain why the resulting protocols are not succinct.

The synthesis procedure has two steps, as visualized in Figure 5.1. In the first step, the procedure constructs one *base* population protocol for each atomic formula of the input formula $\varphi$. These are *threshold formulas* of the form $\sum_{i=1}^{k} a_i x_i < b$ or *modulo formulas* of the form $\sum_{i=1}^{k} a_i x_i \equiv_m b$, where $a_i$, $b$, and $m \geq 2$ are integers constants and $x_i$ is a variable. In the second step, protocols for all atomic formulas are combined with a standard product construction.

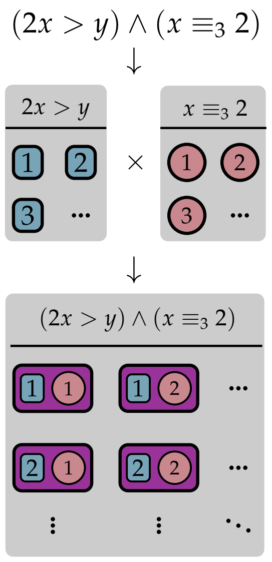

$(2x > y) \wedge (x \equiv_3 2)$
$2x > y$
$x \equiv_3 2$
1
2
3
$\times$
1
2
3
$(2x > y) \wedge (x \equiv_3 2)$
1 1
1 2
2 1
2 2

**Figure 5.1:** Synthesis procedure from [AAD+04] that results in fast but large protocols. In the first step, base protocols for each atomic formula are constructed. Then, the protocols are combined using a product construction.

We will now explain the first step in more detail. The idea of the base protocol construction is similar to the idea in the flock-of-birds protocol (see Example 7): The protocol tries to collect the information about the total value $\sum_{i=1}^{k} a_i x_i$ in a single agent. However, because the number of agents is arbitrary and their memory limited, an agent can only collect value up to a constant $c$. This constant $c$ is chosen high enough to decide if the formula is true. For example, for the threshold formula $3x_1 - 2x_2 < 5$ it is enough to collect a value of $c = 5$ to decide that the formula is satisfied.

**Example 16** (Large Threshold Protocol)**.** *We give a formal description of the protocol from [AAD+04] for a threshold formula* $\sum_{i=1}^{k} a_i x_i < b$*, let* $c \overset{\text{def}}{=} \max(|a_1|, |a_2|, \ldots, |a_k|, |b| + 1)$*.*

*Their protocol for a modulo formula is similar.*

$$Q \overset{def}{=} \{(l, o, v) \mid l \in \{L, F\}, o \in \{\text{true}, \text{false}\}, -c \leq v \leq c\}$$
$$I \overset{def}{=} \{(L, a_i < b, a_i) \mid 1 \leq i \leq k\}$$
$$O((l, o, v)) \overset{def}{=} o$$

*The states of the protocol are triples. Intuitively, the first component indicates whether the agent is a* leader *(L) or a* follower *(F), the second component indicates the agent's current output, and the third component is the value collected by the agent. Agents start as leaders with a value equal to their coefficient in the formula.*

*An interaction between two agents has no effect if both are followers. If at least one of them is a leader, then they change their state according to the following transition:*

$$(*, *, v_1), (*, *, v_2) \mapsto (L, o, v_{sum}), (F, o, v_{rest})$$

*where* $o = \text{true}$ *if* $v_1 + v_2 < b$ *else* false, $v_{sum} = \max(-c, \min(c, v_1 + v_2))$, *and* $v_{rest} = v_1 + v_2 - s$. *Intuitively, this performs a leader election in the first component. This leader stores as much of the combined value as possible* ($v_{sum}$) *and the follower stores the rest* ($v_{rest}$)*. The output of both agents is set according to the leader's value.*

**Speed and Size.** In a careful speed analysis, Angluin *et al.* show that the synthesized protocols are fast [AAD+04]. However, we will now argue that the protocols are large because of two reasons: (i) the inefficient value representation in base protocols and (ii) the product construction for Boolean combination.

(i) **Value Representation:** Consider the formula $\varphi \overset{\text{def}}{=} x < c$. It has length $|\varphi| = \Theta(\log c)$ because the length of the formula depends on the binary representation of $c$. However, the base protocol for $\varphi$ has a state for each value between 0 and $c$, i.e., it has size $\Theta(c) = \Theta(2^{|\varphi|})$. Intuitively, agents store value in *unary* while the length of the formula depends on the *binary* representation of constants.

(ii) **Boolean Combination:** Consider the formula $\varphi \overset{\text{def}}{=} \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_m$ that is a boolean combination of $m$ atomic formulas, where $\varphi_i \overset{\text{def}}{=} x_i < 1$. Because each atomic formula has a constant size, it holds that $m = \Theta(\frac{|\varphi|}{|\varphi_i|}) = \Theta(|\varphi|)$. For each atomic formula $\varphi_i$, the base protocol construction results in a protocol with 20 states. The product construction for $m$ base protocols with 20 states produces a protocol with $20^m = 20^{\Theta(|\varphi|)}$ states. Intuitively, the product construction *multiplies* states of base protocols while the formula *adds* the length of each atomic formula.

## 5.2 The Second Procedure: Succinct but Slow

In our paper [BEG+20], we give a synthesis procedure that produces succinct protocols. Some of the key techniques in the paper are a binary value representation, the use of helpers, as well as the separation of small and large inputs. These techniques will be explained in the subsequent section, where we discuss the state-of-the-art synthesis procedure that incorporates all of them.

While the resulting protocols are succinct, they are also slow. We illustrate this via the (slightly simplified) protocol that is synthesized for the flock-of-birds task.

**Example 17** (A Succinct but Slow Flock-of-Birds Protocol)**.** *Assume that we have a flock with an unknown number of birds. Each bird has a sensor that detects if it is "sick" (1) or "healthy" (0). We will now give a family of population succinct protocols that computes if the number of sick birds is at least $2^d$ for some parameter $d \in \mathbb{N}$.*

$$Q \overset{def}{=} \{0,1,2,4,8\dots,2^{d-1},2^d\} \qquad I \overset{def}{=} \{0,1\} \qquad O(s) \overset{def}{=} (s = 2^d)$$

*Note that there is only a state for every power of two and not for every value. Thus, the number of states is $d+1 = \mathcal{O}(|\varphi|)$ and the protocol is succinct. There are three types of transitions:*

$$\begin{aligned}
up_i : &\quad 2^i, 2^i \mapsto 2^{i+1}, 0 && \textit{for every } 0 \le i \le d \\
down_i : &\quad 2^{i+1}, 0 \mapsto 2^i, 2^i && \textit{for every } 0 \le i < d \\
true_x : &\quad 2^d, x \mapsto 2^d, 2^d && \textit{for every } x \in Q
\end{aligned}$$

*Intuitively, the up-transitions combine the information of two agents with value $2^i$, producing an agent with value $2^{i+1}$ and an agent with value 0. The down-transitions undo up-transitions and allow the splitting of larger powers of two. The true-transitions propagate the information that there are $2^d$ sick birds. For a Petri net visualization of the protocol, see Figure 5.2.*

*The protocol is correct because if there are at least $2^d$ sick birds, then it is always possible to accumulate that value in a single agent via a sequence of up-transitions. At that point, the down-transitions cannot undo this progress and the true-transitions lead to the correct consensus.*[2]

*To understand why the protocol is slow, we will analyze its worst-case input: a flock of $n \gg 2^d$ birds with exactly $2^d$ sick birds and a large number of $n - 2^d$ healthy birds. For the protocol to reach the correct consensus, a single agent needs to collect all information and reach state $2^d$. Until this happens, there always is a large number $\mathcal{O}(n)$ of agents in state 0, but the number of agents with positive value is at most $2^d = \mathcal{O}(1)$, i.e., constant in $n$. Intuitively, this implies that the probability of progress via an up-transition is $\mathcal{O}(\frac{1}{n^2})$, while the probability of regress via a down-transition is $\Omega(\frac{1}{n})$. Using the theory for biased random walks, this tells us that the expected number of interactions is $\Omega(n^{2^d-2})$.*

Please note that the general construction for arbitrary formulas even requires $2^{\Omega(n)}$ interactions (see [CGH+22, Example 7] or Appendix E).

[2] The attentive reader will notice that the *down*-transitions are not necessary for the correctness of this example. They are only present because the general construction for threshold formulas also needs to handle negative values. These negative values need to be able to "cancel out" with positive values. Because only agents with the same absolute value can cancel ($2^i, -2^i \mapsto 0, 0$), this requires that larger powers can decay. For more details, see [CGH+22, Example 6] or Appendix E.

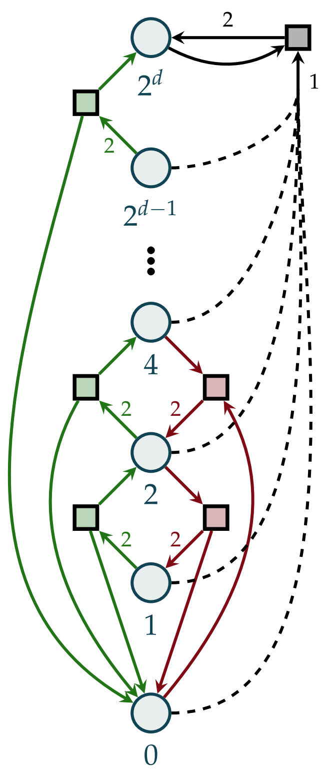


**Figure 5.2:** Petri net visualization of a succinct but slow flock-of-birds protocol. It is a simplified version of the protocol that is synthesized by our procedure in [BEG+20]. All *up*-transitions are colored green and all *down*-transitions are colored red. The *true*-transition is black and consumes one agent in $2^d$ and any other agent as implied by the dashed arcs.

## 5.3 The State-of-The-Art Procedure: Fast and Succinct

In [CGH+22], we give a synthesis procedure that produces fast and succinct population protocols. We will now explain some of its central ideas.

### 5.3.1 Population Computers

The synthesis procedure makes use of a generalization of population protocols called *population computers*. Population computers are easier to design because they extend population protocols in three ways. However, we show that a population computer can be transformed into an equivalent regular population protocol.

First, while in population protocols agents interact in pairs, population computers allow so-called $k$-way interactions, i.e., interactions between $k$ agents.[3] For example, a

[3]Note that $k$-way interactions are possible in chemical reaction networks (see Chapter 6). However, like the 2-way interactions of population protocols, $k$-way interactions do not change the number of agents. This is a key difference to chemical reactions, which can change the number of molecules.

3-way interaction changes the states of three agents and might look like this:

$$A, A, B \mapsto C, D, E$$

The second extension introduces helpers, special auxiliary agents that are present independent of the input. In contrast to the concept of leaders [AAE08], the exact number of helpers is unknown because there might be superfluous helpers. Intuitively, when designing a protocol with leaders, one can rely on the exact number of auxiliary agents and their initial state. However, in a protocol with helpers, we only know that enough helpers are in the desired state but there might be more.

The last extension allows population computers to define the output of a configuration with a more general function. Instead of defining the output of a configuration as its consensus (i.e., as the common opinion of all agents), population protocols can define the output of configurations according to its support. The support of a configuration refers to the set of states that are non-empty. For instance, $\mathrm{supp}(\wr 3 \cdot A, 42 \cdot B \wr) = \{A, B\}$. This extension simplifies the protocol design process, as demonstrated by the following example.

**Example 18** (Population Computer for Majority)**.** *Recall the incorrect majority protocol of Example 5. It has only three of the four transitions of the majority protocol:*

$$t_1 : Y, N \mapsto y, n \qquad t_2 : Y, n \mapsto Y, y \qquad t_3 : N, y \mapsto N, n \qquad \cancel{t_4 : y, n \mapsto y, y}$$

*However, the output function $O(s) \overset{def}{=} (s = Y)$ is unchanged. As explained in Example 5, the protocol does not compute the formula $Y \geq N$ in case of a tie. For example, if we start with $\wr Y, N \wr$, the protocol "gets stuck" in $\wr y, n \wr$, where the agents have different outputs.*

*In contrast, a population computer can leverage the more general output and instead use the following output function:*

$$O(\mathrm{supp}(C)) = \begin{cases} \text{true} & \text{if } \mathrm{supp}(C) \subseteq \{Y, y\} \vee \mathrm{supp}(C) = \{y, n\} \\ \text{false} & \text{if } \mathrm{supp}(C) \subseteq \{N, n\} \end{cases}$$

*Note that this function determines the output of a configuration directly from its support and does not require a consensus of all agents. Because a tie leads to a configuration with support $\{y, n\}$, the population computer with this output function correctly computes majority.*

### 5.3.2 Succinct Population Computers

Our synthesis procedure in [CGH+22] starts by constructing a succinct population computer. While we will not describe the full construction in detail, we will use Figure 5.3 to explain how our approach addresses the two problems encountered in the synthesis procedure of Section 5.1. Please note that Figure 5.3 depicts the full construction, which uses methods that have not been explained at this point. We will clarify the relevant aspects of the visualization where necessary.

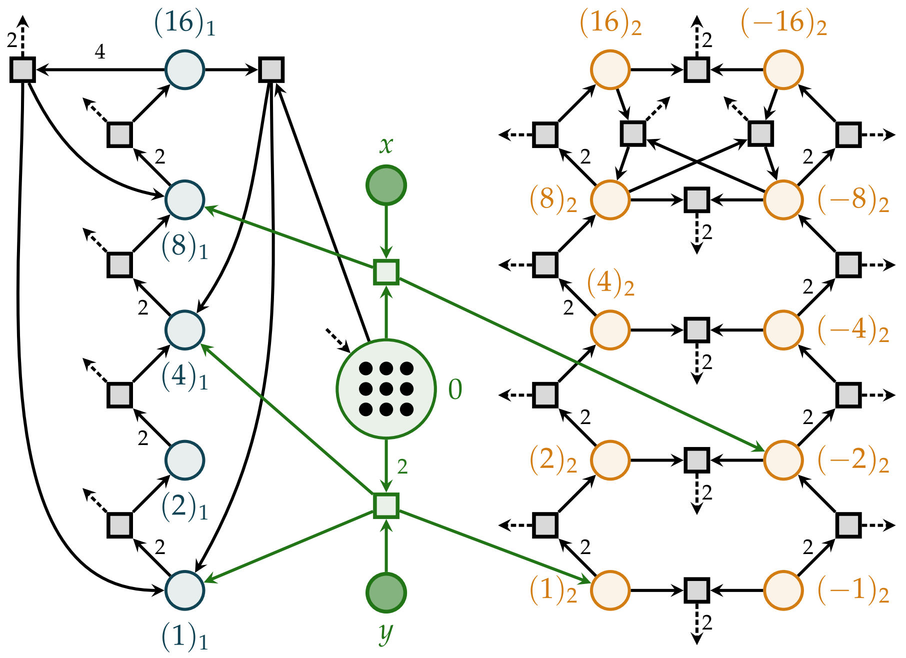


**Figure 5.3:** Petri net visualization of population computer for formula $8x + 5y \equiv_{11} 4 \vee y - 2x \geq 5$. Places (circles), transitions (squares), and tokens (dots) represent states, transitions, and helpers, respectively. All dashed arrows implicitly lead to the reservoir state 0. It has 22 helpers, although only 9 are drawn for space reasons. The computer has three parts: The left part with blue states calculates the modulo formula $8x + 5y \equiv_{11} 4$, the right part with orange states calculates the threshold formula $y - 2x \geq 5$, and the green middle distributes input agents in the initial states $x$ and $y$. For details on the output function, see [CGH+22] or Appendix E.

**Value Representation.** Figure 5.3 shows a population computer for a formula that is a Boolean combination of two atomic formulas. The left half of the visualization corresponds to a modulo formula and the right half to a threshold formula. In both parts of the computer, there is a ladder-like structure of states with increasing value. Note that there are only states with absolute values that are a power of two. Intuitively, this allows to represent value in binary compared to the unary representation in Section 5.1.

**Binary Combination.** The center of Figure 5.3 shows the initial states $x$ and $y$ of formula $8x + 5y \equiv_{11} 4 \vee y - 2x \geq 5$. An "input" agent in one of these states contributes value in both atomic formulas. For example, an agent in state $y$ needs to represent the value 5 in the modulo computation on the left and the value 1 in the threshold computation on the right. Recall that the synthesis procedure by Angluin *et al.*, described in Section 5.1, performed a product construction and allowed input agents to participate simultaneously in the computations of all atomic predicates. As this leads to an exponential number of states, we instead make use of helpers so that each agent is only part of a single computation. For example, the bottom green transition

$$y, 0, 0 \mapsto (4)_1, (1)_1, (1)_2$$

consumes one agent in $y$ and two helpers and adds the value $4 + 1 = 5$ in the left computation and 1 in the right computation. Note that there can be an arbitrary number of input agents that need to be distributed and each distribution consumes helpers. However, we are guaranteed only a constant number of helpers. Thus, the protocol continuously improves the value representation in order to free agents. For example, the transition

$$(2)_1, (2)_1 \mapsto (4)_1, 0$$

combines two agents with value 2 in the left computation to a single agent with combined value 4. This produces an agent with value 0 that can be reused as a helper in future input distributions.

### 5.3.3 Transforming Population Computers into Fast Population Protocols

The succinct population computers explained above are transformed back into regular population protocols in a series of steps. We will only explain the idea for the removal of helpers and refer to [CGH+22] or Appendix E for details and information on the other steps.

Our task is to transform a population computer with helpers into one that does not need helpers. In a population computer without helpers, every agent is an input agent. However, we need to supply additional agents as helpers. The only option is to use some of the input agents as helpers. For this, we make it possible to combine two agents in the same input state $X$:

$$X, X \mapsto 2X, 0$$

Intuitively, the agent in state $2X$ is an input agent in a new input state that represents twice the value of an agent in state $X$. This allows us to use the agent with value 0 as a helper. There is a problem with this approach: In case the number of input agents is small, this construction does not produce enough helpers. This is easy to see if you consider the case where we need 20 helpers but only get 15 input agents. Therefore, our construction treats small inputs separately and performs a backup computation that can exploit the fact that there are few agents (see [BEG+20] or Appendix D).

**Speed.** We show that the resulting population protocols are fast. First, the proof demonstrates the speed $\mathcal{O}(n^3)$ using a layer ranking function (compare Section 4.2.7). Intuitively, this step argues that all non-silent transitions "make progress" and that it is always possible to make progress until the computation is complete. In a second step, we show that progress is not only possible but likely, yielding the speed bound $\mathcal{O}(n^2)$.

We will now illustrate the speed of the resulting protocols via the (slightly simplified) protocol that is synthesized for the flock-of-birds task. Essentially, the protocol is equivalent to the slow protocol of Example 17 (and Figure 5.2) without *down*-transitions:

**Example 19** (A Fast and Succinct Flock-of-Birds Protocol). *Assume that we have a flock with an unknown number of birds. Each bird has a sensor that detects if it is "sick" (1) or "healthy" (0). We will now give a family of fast and succinct population protocols that computes if the number of sick birds is at least $2^d$ for some parameter $d \in \mathbb{N}$.*

$$Q \overset{def}{=} \{0, 1, 2, 4, 8 \ldots, 2^{d-1}, 2^d\} \qquad I \overset{def}{=} \{0, 1\} \qquad O(s) \overset{def}{=} (s = 2^d)$$

*Note that the number of states is $d + 1 = \mathcal{O}(|\varphi|)$, i.e., the protocol is succinct. There are two types of transitions:*

$$\begin{array}{lll} up_i : & 2^i, 2^i \mapsto 2^{i+1}, 0 & \textit{for every } 0 \leq i \leq d \\ true_x : & 2^d, x \mapsto 2^d, 2^d & \textit{for every } x \in Q \end{array}$$

*Intuitively, the up-transitions combine the information of two agents with value $2^i$, producing an agent with value $2^{i+1}$ and an agent with no value. The true-transitions propagate the information that there are $2^d$ sick birds.*

*The protocol is correct because if there are at least $2^d$ sick birds, then it is possible to accumulate that value in a single agent via a sequence of up-transitions. At that point, the true-transitions lead to the correct consensus.*

*To understand why the protocol is fast, we will analyze its worst-case input: a flock of $n \gg 2^d$ birds with exactly $2^d$ sick birds and a large number of $n - 2^d$ healthy birds. For the protocol to reach the right consensus, a single agent needs to collect all information and reach state $2^d$. Until this happens, there always is a pair of agents that can interact via an up-transition. This pair is chosen with probability at least $\frac{1}{n^2}$. As there are $2^d$ sick birds, there will be at most $2^d - 1$ up-transitions. Thus, the expected number of interactions is $\mathcal{O}((2^d - 1)n^2) = \mathcal{O}(n^2)$.*

# 6 Chemical Reaction Networks

In this chapter, we introduce the chemical reaction network (CRN) model. We begin with a high-level explanation to give a first intuition, followed by a comparison to the closely-related model of population protocols from Chapter 3. We give a formal definition of chemical reaction networks, but we refer to [Bri19] for more details.

Chemical reaction networks model the stochastic interactions of molecules that interact according to chemical reactions. They are widely used to model and analyze real-world stochastic systems and have many applications, such as in biochemistry [CBH+09], epidemiology [LMV22], and molecular programming [SSW10]. The model is closely related to population protocols: Similar to agents that change their state according to transitions, *molecules* in chemical reaction networks change *species* via *reactions*.

**Example 20** (A chemical reaction network).

$$2H_2 + O_2 \to 2H_2O$$
$$C + O_2 \to CO_2$$
$$6CO_2 + 6H_2O \to C_6H_{12}O_6 + 6O_2$$

*This chemical reaction network has six species: oxygen $(O_2)$, hydrogen $(H_2)$, water $(H_2O)$, carbon $(C)$, carbon dioxide $(CO_2)$, and glucose $(C_6H_{12}O_6)$. In the first two combustion reactions, oxygen reacts with hydrogen to produce water or with carbon to produce carbon dioxide. The third reaction describes the process of photosynthesis, where water and carbon dioxide are converted to glucose and oxygen.*

Molecules do not need to be described using chemical nomenclature. Instead, they can have abstract names like in the predator-prey system.

**Example 21** (Predator Prey). *The predator-prey chemical reaction network [Gil77] has two species:* Pred *and* Prey. *There are three reactions:*

$$\begin{aligned}
&\textit{rep}: && \textsc{Prey} \xrightarrow{1\cdot\textsc{Prey}} 2\cdot\textsc{Prey}\\
&\textit{eat}: && \textsc{Pred}+\textsc{Prey} \xrightarrow{0.005\cdot\textsc{Pred}\cdot\textsc{Prey}} 2\cdot\textsc{Pred}\\
&\textit{starve}: && \textsc{Pred} \xrightarrow{1\cdot\textsc{Pred}} \emptyset
\end{aligned}$$

*The first reaction allows a* Prey *molecule to reproduce, resulting in two* Prey *molecules. The second reaction describes the predation process, where a predator consumes a prey, leading to a* *decrease in the* PREY *population by one and an increase in the* PRED *population by one. The third reaction models the gradual decline of predators, possibly due to factors such as starvation. Each reaction has a propensity function that determines the speed of the reaction at the current state. We describe its effect below. A simulation of the predator-prey system is shown in Figure 6.1.*

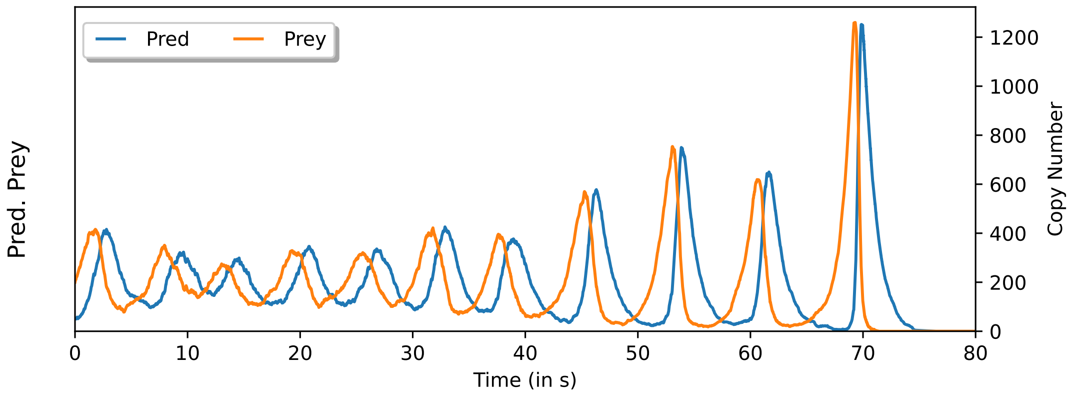


**Figure 6.1:** Simulation of the predator-prey system generated with the stochastic simulation algorithm (SSA). The two species oscillate in an anti-cyclic pattern until one of them dies out.

## 6.1 Comparison with Population Protocols

We will explain the major differences between population protocols and chemical reaction networks.

**Variable Number of Molecules.** While a population protocol works with an arbitrary number of initial agents, this number stays constant during an execution. This is a result of the fact that each transition consumes two agents and produces two agents. In contrast, chemical reactions do not have such a restriction and can have pre- and post-sets of arbitrary size. This makes it possible to increase and decrease the total number of molecules (compare reactions *rep* and *starve*). The amount of molecules can even change by more than one and it is possible to have reactions with empty presets that continuously produce molecules. The global state of a population protocol is referred to as configuration and the number of reachable configurations is always finite. For a chemical reaction network, the global state is simply called its *state*. Due to the potential unbounded growth of the number of molecules, the number of reachable states in a chemical reaction network can be infinite.[1]

[1]Note that in nature, due to physical constraints, the number of molecules is actually bounded if either the volume or the mass of the well-mixed solution is bounded.

**Continuous Time.** In a stochastic population protocol, time is measured in occurred interactions, i.e., the model behaves like a discrete-time Markov chain (DTMC). In contrast, a chemical reaction network uses continuous time, typically measured in seconds, and behaves like a continuous-time Markov chain (CTMC). Specifically, each reaction occurs at a different rate that may vary during an execution and is determined by a *propensity function* $f(s)$ that maps the current state $s$ to a non-negative *propensity*. For example, the propensity function of the reaction

$$\mathit{eat} : \textsc{Pred} + \textsc{Prey} \xrightarrow{0.005 \cdot \textsc{Pred} \cdot \textsc{Prey}} 2\textsc{Pred}$$

is the function $f(s) \overset{\text{def}}{=} 0.005 \cdot s(\textsc{Pred}) \cdot s(\textsc{Prey})$. If the predator-prey system is in state $s \overset{\text{def}}{=} \{\!\!\{4 \times \textsc{Pred}, 20 \times \textsc{Prey}\}\!\!\}$, then the propensity of reaction *eat* is $f(s) = 0.005 \cdot 4 \cdot 20 = 0.4$. The time until an enabled reaction occurs is an independent exponential random variable with a rate equal to the reaction's propensity. In our example, the *eat* reaction is enabled and has a rate of 0.4, i.e., we expect the next *eat* reaction to occur in $\frac{1}{0.4} = 2.5$ seconds.

**Focus on Modeling.** Population protocols are typically designed by humans in order to achieve a goal, i.e., they are programmed to compute some formula. In contrast, chemical reaction networks are primarily used to model stochastic systems with no input or output. Consequently, research on chemical reaction networks typically focuses on efficient techniques that help to predict their transient behavior.

## 6.2 Formal Definition

A CRN $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where $\Lambda$ is a set of *species* and $\mathcal{R}$ is a set of reactions. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^\Lambda$ is the *reactant complex*, $p_\tau \in \mathbb{N}^\Lambda$ is the *product complex*, and $k_\tau \in \mathbb{R}_{>0}$ is the rate constant. We use standard chemical notation to describe a reaction, e.g., a reaction $\tau_1 = (\{\!\!\{A, B\}\!\!\}, \{\!\!\{2 \cdot C\}\!\!\}, 5)$ is written as $\tau_1 : A + B \xrightarrow{5} 2 \cdot C$. The *effect* of $\tau$ is defined as $\Delta(\tau) \overset{\text{def}}{=} p_\tau - r_\tau$. A *state* of CRN $\mathcal{N}$ is a multiset $s \in \mathbb{N}^\Lambda$ that describes the *copy number* $s(X)$ for each species $X \in \Lambda$. A reaction $\tau = (r_\tau, p_\tau, k_\tau)$ can *occur* in $s$ if $s \geq r_\tau$, i.e., if there are enough reactant molecules. This leads to the state $s' = s + \Delta(\tau)$, which we write as $s \xrightarrow{\tau} s'$.

Under the assumption of mass action kinetics, the *propensity function* of a reaction $\tau = (r_\tau, p_\tau, k_\tau)$ is:

$$f_\tau(s) \overset{\text{def}}{=} k_\tau \prod_{X \in \Lambda} \binom{s(X)}{r_\tau(X)}$$

Intuitively, the propensity counts the number of ways one can pick all molecules of the reactant complex $r$ from the available molecules in state $s$, and the rate constant $k_\tau$ represents the likelihood that the reaction occurs if they all meet. For other kinetics, such as Michaelis-Menten and Hill, the propensity function can be defined differently.

Thus, we usually give the full propensity function when describing a reaction, e.g., for reaction $\tau_1 = (\wr A, B\wr, \wr 2 \cdot C\wr, 5)$ we write $\tau_1 : A + B \xrightarrow{5 \cdot A \cdot B} 2 \cdot C$.

The time-evolution of a chemical reaction network is governed by the Chemical Master Equation (see [Gil92]) that leads to a (potentially infinite) discrete-space, continuous-time Markov chain (CTMC) $\mathbf{X}(t) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t))_{t \geq 0}$. This CTMC describes how the probability of the copy numbers of each species evolve in time. The *transition rate* $f(s, s')$ from state $s$ to state $s'$ in the CTMC is the sum of propensities of reactions that lead from $s$ to $s'$:

$$f(s, s') \overset{\text{def}}{=} \sum_{\substack{\tau \in \mathcal{R} \\ s \xrightarrow{\tau} s'}} f_\tau(s)$$

As we do not assume familiarity with CTMCs, we will now give a short explanation of how this system behaves. The time until the next reaction occurs is an exponential random variable with rate

$$f(s) \overset{\text{def}}{=} \sum_{\tau \in \mathcal{R}} f_\tau(s)$$

This continuous random variable depends only on the current state and not on the amount of time elapsed, i.e., it has no memory. The expected time for the next reaction to occur is thus $1/f(s)$. The probability that reaction $\tau \in \mathcal{R}$ is the next reaction to occur in $s$ is $f_\tau(s)/f(s)$.

**Example 22** (CTMC for Predator Prey). *Recall the three reactions of the predator-prey system:*

$$\begin{aligned}
&\textit{rep}: && \textsc{Prey} \xrightarrow{1 \cdot \textsc{Prey}} 2 \cdot \textsc{Prey} \\
&\textit{eat}: && \textsc{Pred} + \textsc{Prey} \xrightarrow{0.005 \cdot \textsc{Pred} \cdot \textsc{Prey}} 2 \cdot \textsc{Pred} \\
&\textit{starve}: && \textsc{Pred} \xrightarrow{1 \cdot \textsc{Pred}} \emptyset
\end{aligned}$$

*If the system is in state* $s \overset{def}{=} \wr 4 \times \textsc{Pred}, 20 \times \textsc{Prey}\wr$, *then all three reactions can occur and their propensities are:*

$$\begin{aligned}
f_{rep}(s) &= 1 \cdot s(\textsc{Prey}) = 20 \\
f_{eat}(s) &= 0.005 \cdot s(\textsc{Pred}) \cdot s(\textsc{Prey}) = 0.005 \cdot 4 \cdot 20 = 0.4 \\
f_{starve}(s) &= 1 \cdot s(\textsc{Pred}) = 4
\end{aligned}$$

*The total propensity is* $20 + 4 + 0.4 = 24.4$, *i.e., we expect the next reaction to occur in* $1/24.4 \approx 0.041$ *seconds. The probabilities for the next reaction are* $20/24.4 \approx 82\%$ *for "rep",* $0.4/24.4 \approx 1.6\%$ *for "eat", and* $4/24.4 \approx 16.4\%$ *for "starve".*

## 6.3 Stochastic Simulation

Gillespie's widely used stochastic simulation algorithm (SSA) [Gil77] produces statistically correct trajectories of the given CRN, i.e., sampled according to the underlying CTMC. As shown in Algorithm 3, SSA repeatedly applies one reaction at a time while keeping track of the elapsed time. Intuitively, each step of the algorithm is similar to the process described in Example 22.

**Algorithm 3** Gillespie's (direct) stochastic simulation algorithm

```
function SIMULATE($state_{\text{initial}}$, $time_{\text{end}}$)
    $s := state_{\text{initial}}$
    $time := 0$
    while $time < time_{\text{end}}$ do
        evaluate propensities $f_\tau(s)$ for each $\tau \in \mathcal{R}$ and compute their sum $f(s)$
        if $f(s) = 0$ then                                    ▷ No reaction can occur
            $time := time_{\text{end}}$
        else
            sample $\Delta t$ using exponential distribution with rate $f(s)$
            $time := time + \Delta t$
            choose reaction $\tau$ with probability $f_\tau(s)/f(s)$
            $s := s - r_\tau + p_\tau$
    return $(s, time)$
```

### 6.3.1 Approximate Stochastic Simulation

A common way to speed up the simulation process is to apply multiple reactions at once. For example, the $\tau$-leaping approach [Gil01] assumes that propensities do not change significantly within the time window $\tau$. This makes it possible to sample the number of occurrences $o_\tau$ of each reaction $\tau$ using a Poisson distribution (see Algorithm 4). It is important to note that $\tau$-leaping is an approximate simulation technique because it does not sample the underlying CTMC exactly. See Section 7.7 for an overview of approximate simulation approaches.

**Algorithm 4** $\tau$-leaping algorithm

```
function SIMULATE($state_{\text{initial}}$, $time_{\text{end}}$, $\tau$)
    $s := state_{\text{initial}}$
    $time := 0$
    while $time < time_{\text{end}}$ do
        evaluate propensities $f_\tau(s)$ for each $\tau \in \mathcal{R}$ and compute their sum $f(s)$
        if $f(s) == 0$ then                                ▷ No reaction can occur
            $time := time_{\text{end}}$
        else
            for all $\tau \in \mathcal{R}$ do
                sample $o_\tau$ using Poisson distribution with rate $\tau \cdot f_\tau(s)$
                $state_{\text{next}} := state_{\text{next}} + o_\tau \cdot \Delta(\tau)$
            $time := time + \tau$
    return $(s, time)$
```

## 6.4 Deterministic Simulation

While stochastic simulations try to capture the random fluctuations and discrete nature of biological systems, deterministic simulations instead neglect any noise and treat species as continuous variables. In this case, the CRN is evolved according to a set of ordinary differential equations (ODEs). The equations specify the derivative of a species $X \in \Lambda$ as follows:

$$\frac{dX}{dt} = \sum_{\tau \in \mathcal{R}} f_\tau \cdot \Delta(\tau)(X)$$

While a deterministic simulation can be very efficient, it often does not capture the behavior of the analyzed system, especially in systems with low copy numbers or multistability. Deterministic simulation is most beneficial when all copy numbers and the impact of random fluctuation is minimal.

**Example 23** (ODEs for Predator Prey)**.** *Recall the reactions of the predator-prey system:*

$$\begin{aligned}
&rep: && \textsc{Prey} \xrightarrow{1\cdot\textsc{Prey}} 2\cdot\textsc{Prey} \\
&eat: && \textsc{Pred} + \textsc{Prey} \xrightarrow{0.005\cdot\textsc{Pred}\cdot\textsc{Prey}} 2\cdot\textsc{Pred} \\
&starve: && \textsc{Pred} \xrightarrow{1\cdot\textsc{Pred}} \emptyset
\end{aligned}$$

*Under the assumption of no noise and continuous copy numbers, the system evolves according to the following set of ordinary differential equations:*

$$\begin{aligned}
\frac{d\textsc{Pred}}{dt} &= 0.005\cdot\textsc{Pred}\cdot\textsc{Prey} - \textsc{Pred} \\
\frac{d\textsc{Prey}}{dt} &= \textsc{Prey} - 0.005\cdot\textsc{Pred}\cdot\textsc{Prey}
\end{aligned}$$

*A deterministic simulation of the system is shown in Figure 6.2. It oscillates indefinitely and does not reflect that the system can die out like in the SSA simulation of Figure 6.1.*

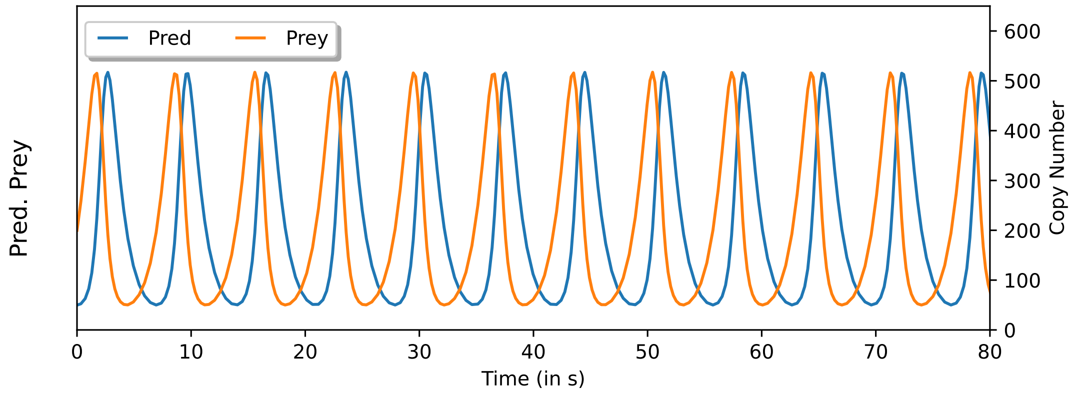


**Figure 6.2:** Deterministic simulation of the predator-prey system generated by solving a system of ordinary differential equations (ODEs). The two species oscillate indefinitely in an anti-cyclic pattern. This is not consistent with the accurate simulation in Figure 6.1.

## 6.5 Simulation-Based Analysis

By performing a large number of stochastic simulations, one can estimate properties of a chemical reaction network. The process is best illustrated by the following example.

**Example 24** (Transient Analysis of Predator Prey)**.** *Consider the predator-prey system of Example 21 starting in initial state* $\wr 200 \cdot \text{PRED}, 200 \cdot \text{PREY} \wr$*. The two species of the system oscillate until one of the species dies out. In case the* PREY *species dies first (because the last* PREY *got eaten), the* PRED *will starve and also die out and the system stays in* $\wr\wr$*. A simulation-based analysis could run 100,000 simulations using SSA for 200 seconds. In approximately 36,000 simulations (or 36% of simulations), the system dies out because the last* PREY *was eaten. Or in other words, if the system starts in* $\wr 200 \cdot \text{PRED}, 200 \cdot \text{PREY} \wr$*, then more than every third evolution dies within the first 200 seconds because the last* PREY *was eaten.*

# 7 Segmental Simulation of Chemical Reaction Networks

In this chapter, we showcase our contributions to the field of chemical reaction network analysis. It is worth noting that in Chapter 4, we explored the realm of population protocol verification, encompassing both theoretical advancements and their practical application. However, as chemical reaction networks primarily serve as models for stochastic systems, our research in this chapter predominantly focuses on the efficient prediction of system properties in practical settings.

A meaningful simulation-based analysis of chemical reaction networks requires a large number of stochastic simulations. This can take numerous hours [HGK15], especially if the generation of a single simulation is slow. To make the simulation-based analysis of chemical reaction networks more efficient, we introduce a novel approximate simulation technique called *segmental simulation*. The fundamental principles underlying segmental simulation are as follows:

- **Acceleration:** Instead of applying one reaction at a time, like in SSA (see Section 6.3), we apply multiple reactions at once by sampling a short trajectory called a *segment*.
- **Memoization[1]:** To efficiently sample a segment for the current state of the system, we reuse parts of previously generated SSA simulations.
- **Abstraction:** Simulations rarely visit the exact same state multiple times but behave similarly if the copy numbers of all species are similar. Thus, we split the state space into regions, called *abstract states*, and reuse a segment that starts in the current region.

Figure 7.1 visualizes how segmental simulation works. On the left, we see a representation of the memory. For each of the four abstract states, it contains three segments. Each segment starts at the same concrete state in the abstract state's center, called the *representative*, and ends when it leaves the abstract state. These segments were previously generated via SSA and effectively approximate the local dynamics of the system. On the right of Figure 7.1, we see a segmental simulation starting at concrete state $s_{init}$. It is generated by repeatedly choosing one of the saved segments for the current abstract states and applying its effect to the current concrete state. Note that the segment also has an effect on the elapsed time that is not visualized.

[1] As mentioned in the introduction, *memoization* is an optimization technique that stores results to avoid recalculation, while *memorization* (with 'r') is the process of committing something to memory.

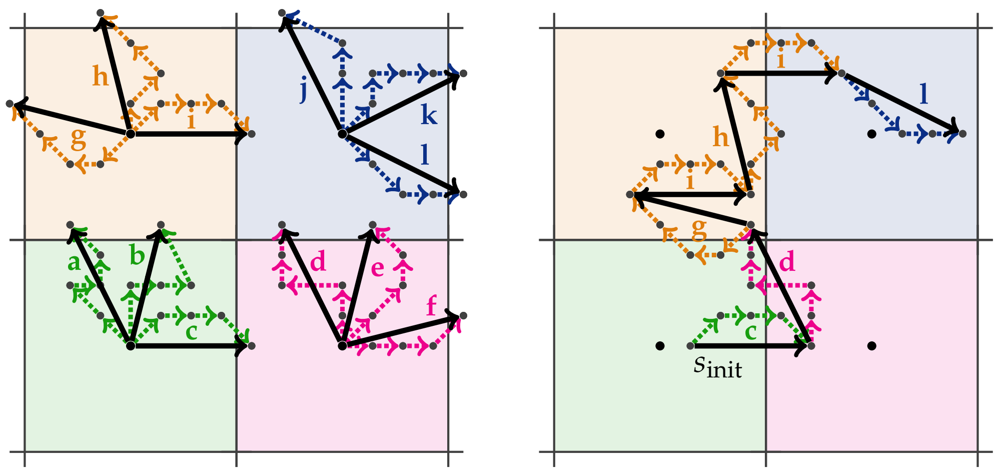


**Figure 7.1:** Visualization of segmental simulation algorithm. (left) Four neighboring abstract states are drawn as squares. Each abstract state is displayed with three segments that start in their respective centers. Each segment is a sequence of reactions drawn as dotted arrows. The difference between the endpoint and the starting point is called a *summary* and is drawn in unbroken black. (right) A possible segmental simulation is obtained by applying segments $c, d, g, i, h, i$, and $l$ to the initial state $s_{\text{init}}$.

Algorithm 5 summarizes the basic scheme of abstraction-based segmental simulation. It reuses and fills the *memory*, which potentially already contains segments that were generated in previous simulations. As we simulate, we always compute the current abstract state $a$ (Line 4) and randomly choose which of the $k$ segments for $a$ will be reused (Line 5). If the segment was already computed, we load it from memory (Line 7). Otherwise, we generate a new segment starting at $a$'s representative $r$ (Line 10) and store it. Finally, we apply the chosen segment in Lines 12 and 13. Because the segment may not start at the current state, we only apply the segment's total effect on state and time. To reduce memory consumption, we typically only save the effect of a segment which we call its *summary*.

The following sections give more details about the segmental simulation approach. We explain the population-based abstraction we use to divide the state space into abstract states in Section 7.1. Section 7.2 contains a theoretical discussion of the two error sources of this approximate simulation approach. We generalize segmental simulation to improve both memory usage and performance in Section 7.3. In Section 7.4, we introduce a new fully automatic hybrid simulation scheme that can be combined with segmental simulation. An evaluation of the approach is given in Section 7.5. We introduce SAQuaiA, a free and easy-to-use tool that implements segmental simulation, in Section 7.6. Finally, Sections 7.7 and 7.8 give an overview of related work and open research questions.

**Algorithm 5** Segmental Simulation

```
Require: memory (mapping from abstract state to list of segments; can contain data
    from previous simulations)
 1: function SIMULATE(state, $time_{\text{end}}$)
 2:     $time := 0$
 3:     while $time < time_{\text{end}}$ do
 4:         $a := \text{ABSTRACTSTATE}(state)$
 5:         $n := \text{RANDOM}(\{1, .., k\})$                          ▷ Choose segment
 6:         if $n \leq |memory(a)|$ then                                ▷ Already exists?
 7:             $segment := memory(a).\text{GET}(n)$                    ▷ Reuse
 8:         else
 9:             $r := \text{REPRESENTATIVE}(a)$                          ▷ Lazy generation
10:             $segment := \text{NEWSSASEGMENTFROM}(r)$
11:             $memory(a).\text{ADD}(segment)$
12:         $state := state + segment.\Delta_{\text{state}}$             ▷ Apply segment's effect
13:         $time := time + segment.\Delta_{\text{time}}$
14:     return $(state, time)$
```

## 7.1 Population-Level Abstraction

We will now explain the abstraction we use to divide the state space of the chemical reaction network into regions we call *abstract states*. While a general abstraction can be arbitrarily complex, we use a *population-level abstraction* that defines population levels by partitioning every dimension into intervals. Effectively, this divides the state space into hyper-rectangles.

Specifically, a population-level abstraction defines consecutive intervals for each dimension. Each interval is annotated with a representative that is part of the interval. We write $[x, y, z]$ with $x \leq y \leq z$ to denote the interval containing the values $\{x, x+1, ..., z-1, z\}$ with representative $y$ (e.g., $[3, 4, 6] = \{3, 4, 5, 6\}$ with representative 4). An abstract state in a $d$-dimensional system is a hyper-rectangle that is described by $d$ intervals, one for each dimension. The representative of an abstract state is the concrete state that corresponds to the representatives of all its intervals. For example, the abstract state in a two-dimensional system for intervals $[3, 4, 6]$ and $[11, 14, 18]$ contains all concrete states $(x, y)$ with $3 \leq x \leq 6$ and $11 \leq y \leq 18$ and has representative $(4, 14)$.

The abstraction function for a given population-level abstraction maps each concrete state to its abstract state denoted as the vector of levels. E.g., for the interval abstraction in Table 7.1, the concrete state $(16, 269)$ is mapped to abstract state $(4, 10)$ because 16 is in level 4 and 269 is in level 10.

**Abstraction Suitable for Segmental Simulation.** While in principle, one can use any population-level abstraction for segmental simulation, inconsistencies such as negative copy numbers, jumps over abstract states, and the application of disabled reactions can

**Table 7.1:** Population-level abstraction for the predator-prey model. Note that this abstraction uses the same intervals for both species, while in general, each dimension can have different population levels.

| level | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| min | 0 | 1 | 3 | 6 | 11 | 19 | 31 | 49 | 76 | 117 | 179 | 272 | 442 | 622 | 937 |
| rep | 0 | 1 | 4 | 8 | 14 | 24 | 39 | 62 | 96 | 147 | 225 | 341 | 516 | 779 | 1173 |
| max | 0 | 2 | 5 | 10 | 18 | 30 | 48 | 75 | 116 | 178 | 271 | 441 | 621 | 936 | 1409 |

arise if we do not choose the abstraction carefully. Example 25 highlights all three of these issues.

**Example 25** (Inconsistencies Because of Unsuitable Abstraction)**.** *Consider the system visualized in Figure 7.2 with a single reaction* $r : 2X \to \emptyset$ *and the partitioning into three abstract states* $a_1 = [0,0,1]$, $a_2 = [2,3,4]$ *and* $a_3 = [5,10,\infty]$. *The only possible segment starting at the representative of* $a_3$ *is the sequence* $10 \xrightarrow{r} 8 \xrightarrow{r} 6 \xrightarrow{r} 4$ *leading to abstract state* $a_2$. *However, when we apply this segment to the concrete state 5 of the same abstract state, we get* $5 \xrightarrow{r} 3 \xrightarrow{r} 1 \xrightarrow{r} -1$. *This is incorrect, as negative copy numbers are reached. Furthermore, when applying the same segment to the concrete state 7, we get* $7 \xrightarrow{r} 5 \xrightarrow{r} 3 \xrightarrow{r} 1$ *reaching abstract state* $a_1$. *Although this is a feasible sequence of the system, this segmental step jumped over the abstract state* $a_2$*, effectively ignoring the local dynamics of* $a_2$.

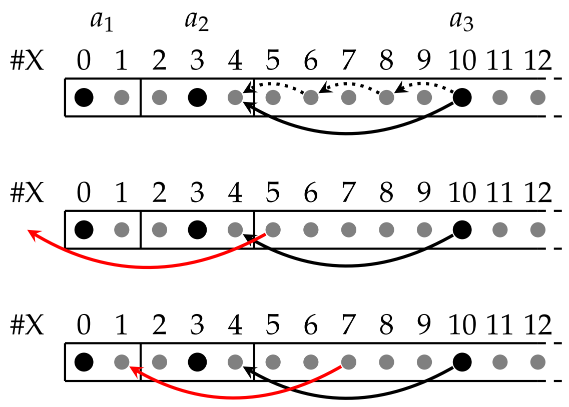


**Figure 7.2:** Problematic segmental simulation steps because of an unsuitable abstraction with three abstract states $a_1, a_2, a_3$. Circles are concrete states. The representatives are bigger and black. (top) The solid arrow is a segment for $a_3$ consisting of three reactions drawn as dotted arrows. (middle) Applying the segment to concrete state 5 leads to a negative copy number. (bottom) Applying the segment to concrete state 7 jumps over $a_2$.

To make sure that no inconsistencies occur, the abstraction used for segmental simulation needs to satisfy some additional properties.

A population-level abstraction is considered *suitable for segmental simulation* if applying any segment of a representative to any concrete state within the same abstract state

1. does not apply reactions that are disabled, and
2. does not lead to a non-neighboring abstract state.

Note, Property (1) implies that the same set of reactions is enabled in all concrete states of an abstract state and negative copy numbers cannot be reached. Further, to ensure that segmental simulation adequately approximates the dynamics of the system, it must hold that the states within each abstract state emit a similar probability space of the trajectories. Since propensity functions are continuous, this assumption holds for sufficiently large abstract states [MMR+12; AAČ+21].

**Exponential Population-Level Abstraction.** In this work, we specifically employ the *exponential population-level abstraction*, which is suitable for segmental simulation. This abstraction uses intervals with sizes that grow exponentially. This results in small intervals when the copy number is low and fluctuations play a crucial role. Conversely, when the number of molecules is high and a single reaction has a relatively lower impact, the exponential growth results in large intervals. To control the rate of interval growth, we utilize a parameter $c$, referred to as the *growth factor*.

Let $\mathcal{N} = (\Lambda, \mathcal{R})$ be a CRN and $c \in \mathbb{R}_{\geq 1}$ be the growth factor. For each species $s \in \Lambda$, we first compute $m_s = \max_{\tau \in \mathcal{R}}(r_\tau)$ the highest multiplicity of $s$ in the reactant complex of any reaction. If $s$ does not react, i.e., $m_s = 0$, then we do not split the dimension of $s$ into multiple segments as the number of $s$-molecules is not important for reusing segments.[2] Otherwise, we add the intervals $[0,0,0]$, $[1,1,1]$,..., $[m_s-1, m_s-1, m_s-1]$ and define the following intervals iteratively: After the interval $i = [x,y,z]$ with $x \leq y \leq z$ and size $|i| := z - x + 1$, we add the interval $i' = [x', y', z']$ where $x' := z + 1$, $y' := z + |i|$, and $z' := \lceil c \cdot |i| \rceil$. Intuitively, the next interval starts after the previous interval, has the desired size of $\lceil c \cdot |i| \rceil$, and its representative is the largest value that does not allow jumps over the previous interval. Please note that the abstraction in Table 7.1 is an exponential population-level abstraction with growth factor $c = 1.5$. In case we want to force additional user-defined levels (e.g., for more precision in a certain range of copy numbers), we can generate intervals using a similar (but more complicated) heuristic or with a constraint solver.

## 7.2 Theoretical Accuracy

Segmental simulation has the following two error sources.

[2] For some applications, like measuring the accuracy in the abstract domain, it can be important to force a partition by setting $m_s := \max(m_s, 1)$.

**Segment Distribution Approximation Error.** By reusing a finite number of saved segments, we effectively sample an underapproximation of the actual segment distribution starting at the representative of an abstract state. If the number of saved segments is too small, the abstract might miss important local behavior or skew the probabilities of events. However, the error decreases when the number of segments is increased and vanishes in the limit.

**Abstraction error.** Recall that segmental simulation does not sample the segment distribution for the current state but instead samples the distribution for the representative of the current abstract state. Because the propensities and thus the rates of reactions are different for different states, this inherently introduces an error. The abstraction error is reasonably small in practice as the segment distributions for states within one abstract state are quite similar: Consider that within any abstract state, the propensity and thus rate for a mass-action reaction varies by at most the factor $c^r$ where $c$ is the growth factor of the exponential abstraction and $r$ is the number of reactants. Decreasing the size of abstract states decreases the abstract error, and it vanishes for $c{=}1$, where every state corresponds to a different abstract state.

## 7.3 Generalized Segmental Simulation

Segmental simulation has two major limitations.

1. The *memory* requirements can be large, especially for systems with many dimensions.
2. The generation of segments may take a prohibitively large amount of *time*, e.g., if the copy numbers are large and the segments consist of many reactions.

We generalize the basic segmental simulation approach as shown in Algorithm 6 to overcome these limitations.

**Improving Memory Consumption.** The memoization is managed by a general *artifact* that approximates the segment distribution of abstract states. This makes it possible to use more memory-efficient segment distribution approximations, such as the approximation explained by Figure 7.3. Furthermore, the artifact can dynamically adjust how the available memory is used. For each segmental simulation step, there are three options:

1. The artifact can decide not to use memoization and instead evolve the system normally (Line 6). This is useful if the memory is full or memoization for the current abstract state is inefficient.
2. The artifact can decide to improve the segment distribution approximation of the current abstract state by generating a new segment starting at the representative (Line 8). This allows us to dynamically improve the local precision, e.g., if an abstract state is visited a lot.

**Algorithm 6** Generalized Segmental Simulation

**Require:** *artifact* (mapping from abstract state to segment distribution approximation; can contain data from previous simulations), *base* (simulation technique for generating new segments)

```
function SIMULATE($state$, $time_{\text{end}}$)
    $time := 0$
    while $time < time_{\text{end}}$ do
        $a :=$ ABSTRACTSTATE($state$)
        if $artifact$.SHOULDIGNORE($a$) then
            $segment := base$.NEWSEGMENTFROM($state$)          ▷ No memoization
        else if $artifact$.SHOULDIMPROVE($a$) then
            $r :=$ REPRESENTATIVE($a$)                         ▷ Improve local precision
            $segment := base$.NEWSEGMENTFROM($r$)
            $artifact$.IMPROVEAPPROXIMATION($a, segment$)
        else
            $segment := artifact$.SAMPLEAPPROXIMATIONOF($a$)   ▷ Speed up
        $state := state + segment.\Delta_{\text{state}}$        ▷ Apply segment
        $time := time + segment.\Delta_{\text{time}}$
    return $(state, time)$
```

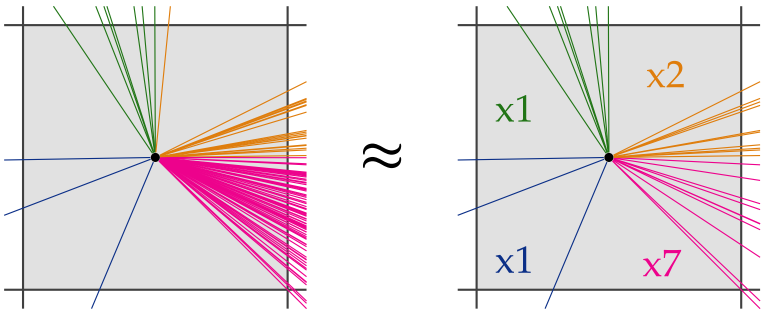


**Figure 7.3:** A more memory-efficient segment distribution approximation. (left) Approximation of a segment distribution made up of 100 summaries. All summaries with the same direction have the same color, e.g., there are 20 orange summaries in direction $(+1, +1)$ or northeast. (right) A very similar but more memory-efficient approximation with 30 summaries. For directions with many summaries, all but ten random summaries were discarded. In order to keep the distribution similar, the weights of those directions are increased. For example, in the northeast direction, the number of summaries was halved but their weight doubled. Thus, choosing a segment in this direction is still equally likely.

3. The artifact can speed up the simulation by sampling from the segment distribution approximation of the current abstract state (Line 12).

An implementation of segmental simulation can dynamically decide for one of these options by tracking statistics about each abstract state. These can include the memory consumption of the segment distribution approximation, the expected speedup of using memoization in this region, and the frequency with which simulations visit the abstract state.

**Improving Performance.** Instead of generating new segments via SSA, Algorithm 6 uses an abstract base simulator. This makes it possible to combine segmental simulation with other approximate simulation methods, such as $\tau$-leaping (see Section 6.3.1) or hybrid simulation (see Section 7.7), in order to achieve even better performance.

## 7.4 Abstraction-Based Hybrid Simulation

We introduce a new hybrid simulation approach that seamlessly combines with segmental simulation and allows to produce segments faster than SSA. It distinguishes three reaction speeds which are evolved using different techniques: slow reactions are simulated via SSA, medium reactions are simulated using $\tau$-leaping and fast reactions are treated as continuous and deterministic, allowing us to evolve them deterministically according to the underlying ODEs.

**Reaction Classification.** The speed classification cannot be done well a priori for the whole system, since the copy numbers of species can vary a lot within a single simulation. Therefore, the classification should depend on the actual state [HGK15]. Our fully automatic reaction classification is done per abstract state. First, each species is assigned a target classification according to the size of the interval in the corresponding dimension. Typically, if the interval of a species is larger than 400, we classify as fast; if it is only larger than 5, we classify as medium; and otherwise as slow. Then, we classify each reaction according to the slowest species affected by the reaction. For example, if the predator-prey system is in the abstract state with PRED interval $[937, 1173, 1409]$ and PREY interval $[19, 24, 30]$, the target speed for PRED is fast and the target speed for PREY is medium. The reaction $\text{PRED} \xrightarrow{1\cdot\text{PRED}} \emptyset$ affects only PRED and thus has a fast speed. The reaction $\text{PRED} + \text{PREY} \xrightarrow{0.005\cdot\text{PRED}\cdot\text{PREY}} 2\text{PRED}$ affects both species and thus has speed medium.

**Hybrid Step.** To perform one step of the hybrid simulation, we first determine the time $\Delta t = \min(\Delta t_{\text{slow}}, \Delta t_{\text{medium}}, \Delta t_{\text{fast}})$ for the next hybrid simulation step. Here, $\Delta t_{\text{slow}}$ is the time of the next slow reaction according to SSA, $\Delta t_{\text{medium}} := \tau$ according to the $\tau$-leaping approach of [CGP06], and $\Delta t_{fast}$ is the time needed for the fast reactions to change the abstract state according to the underlying ODEs (compare Section 6.4). Next,

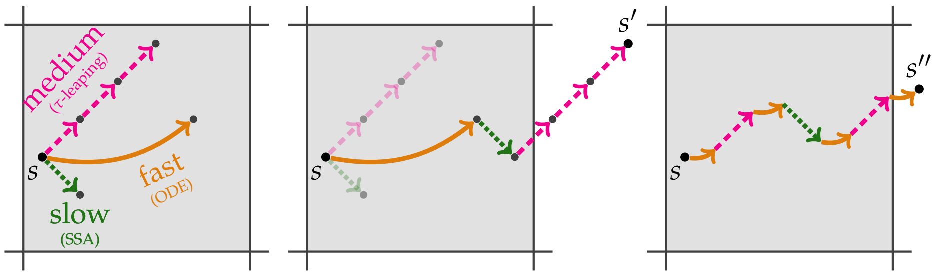


**Figure 7.4:** An abstraction-based hybrid simulation step starting in state $s$ of an abstract state. (left) For every speed, the effects are first calculated separately. (middle) Then, the effects are combined, potentially leading out of the abstract state to state $s'$. (right) As the combined effect was too large, the events are replayed in random order. Between every discrete reaction, there is a continuous evolution. The first state outside of the abstract state is $s'' \neq s'$.

we evolve the state according to the ODEs for fast reactions up to time $\Delta t$. Finally, we sample the discrete reactions that occur in this step and apply their effect. The number of occurrences for medium reactions is determined by $\tau$-leaping for $\tau := \Delta t$. In case $\Delta t < \Delta t_{\mathrm{SSA}}$, the next slow reaction is too late and ignored. Otherwise, the next slow reaction occurs and is sampled according to SSA.[3] This process is illustrated in Figure 7.4.

**Overshooting.** Our hybrid simulation needs to handle cases where a single hybrid step is significantly larger than expected. This is typical whenever $\tau$-leaping is used, as it is unlikely but possible to sample very large values from the Poisson distributions. This could lead to negative copy numbers. A common solution for these rare events in $\tau$-leaping is to discard the step and retry with a smaller $\tau$. In our hybrid simulation approach, we reclassify once we leave the abstract state. Thus, we also consider steps too large if they do not stop right after the abstract state border. As this is quite common, we do not discard the sampled step but replay all reactions in random order until we have left the abstract state (see the right part of Figure 7.4).

## 7.5 Evaluation

We will evaluate three approximate simulation approaches: abstraction-based hybrid simulation (HYB), segmental simulation (SEG), and their combination hybrid segmental simulation (HYBSEG). As case studies, we use the following models from the literature: (1.) predator prey (PP), a.k.a. Lotka-Volterra [Gil77], (2.) repressilator (RP) [HGK15], (3.) toggle switch (TS) [HGK15], and (4.) viral infection (VI) [SYS+02]. Table 7.2 gives a formal definition of each model. The models are challenging due to stochasticity, multi-scale species populations, stiffness, and/or complicated transient

[3]The interaction between slow and medium reactions is analogous to critical and non-critical reactions in [CGP06].

**Table 7.2:** Definitions of models predator prey (PP), toggle switch (TS), repressilator (RP), and viral infection (VI).

| **Predator Prey (PP)** | |
|---|---|
| Species (2) | PRED, PREY |
| Initial state | $(200 \times \mathrm{PRED}, 200 \times \mathrm{PREY})$ |
| End time | $200s$ |
| Reactions<br>(3) | $rep : \mathrm{PREY} \xrightarrow{1\cdot \mathrm{PREY}} 2 \cdot \mathrm{PREY}$<br>$eat : \mathrm{PRED} + \mathrm{PREY} \xrightarrow{0.005\cdot \mathrm{PRED}\cdot \mathrm{PREY}} 2 \cdot \mathrm{PRED}$<br>$starve : \mathrm{PRED} \xrightarrow{1\cdot \mathrm{PRED}} \emptyset$ |

| **Toggle Switch (TS)** | | |
|---|---|---|
| Species (6) | MA, MB, SA, SB, PA, PB | |
| Initial state | $\emptyset$ | |
| End time | $50000s$ | |
| Reactions<br>(14) | $r0 : \emptyset \xrightarrow{1} \mathrm{MA}$<br>$r1 : \emptyset \xrightarrow{1} \mathrm{MB}$<br>$r2 : \mathrm{MA} \xrightarrow{0.1\cdot \mathrm{MA}} \emptyset$<br>$r3 : \mathrm{MB} \xrightarrow{0.1\cdot \mathrm{MB}} \emptyset$<br>$r4 : \mathrm{PA} \xrightarrow{0.1\cdot \mathrm{PA}} \emptyset$<br>$r5 : \mathrm{MA} \xrightarrow{5\cdot \mathrm{MA}} \mathrm{SA}$<br>$r6 : \mathrm{MB} \xrightarrow{5\cdot \mathrm{MB}} \mathrm{SB}$ | $r7 : \mathrm{MB} + \mathrm{SA} \xrightarrow{20\cdot \mathrm{MB}\cdot \mathrm{SA}} \mathrm{SA}$<br>$r8 : \mathrm{MA} + \mathrm{SB} \xrightarrow{20\cdot \mathrm{MA}\cdot \mathrm{SB}} \mathrm{SB}$<br>$r9 : \mathrm{PB} \xrightarrow{0.1\cdot \mathrm{PB}} \emptyset$<br>$r10 : \mathrm{SA} \xrightarrow{0.01\cdot \mathrm{SA}} \emptyset$<br>$r11 : \mathrm{SB} \xrightarrow{0.01\cdot \mathrm{SB}} \emptyset$<br>$r12 : \mathrm{SA} \xrightarrow{10\cdot \mathrm{SA}} \mathrm{SA} + \mathrm{PA}$<br>$r13 : \mathrm{SB} \xrightarrow{10\cdot \mathrm{SB}} \mathrm{SB} + \mathrm{PB}$ |

| **Repressilator (RP)** | | |
|---|---|---|
| Species (6) | MA, MB, MC, PA, PB, PC | |
| Initial state | $(10 \times \mathrm{MA}, 500 \times \mathrm{PA})$ | |
| End time | $50000s$ | |
| Reactions<br>(15) | $spawnA : \emptyset \xrightarrow{0.1} \mathrm{MA}$<br>$spawnB : \emptyset \xrightarrow{0.1} \mathrm{MB}$<br>$spawnC : \emptyset \xrightarrow{0.1} \mathrm{MC}$<br>$prodA : \mathrm{MA} \xrightarrow{50\cdot \mathrm{MA}} \mathrm{MA} + \mathrm{PA}$<br>$prodB : \mathrm{MB} \xrightarrow{50\cdot \mathrm{MB}} \mathrm{MB} + \mathrm{PB}$<br>$prodC : \mathrm{MC} \xrightarrow{50\cdot \mathrm{MC}} \mathrm{MC} + \mathrm{PC}$<br>$despawnA : \mathrm{MA} \xrightarrow{0.01\cdot \mathrm{MA}} \emptyset$<br>$despawnB : \mathrm{MB} \xrightarrow{0.01\cdot \mathrm{MB}} \emptyset$ | $despawnC : \mathrm{MC} \xrightarrow{0.01\cdot \mathrm{MC}} \emptyset$<br>$degradeA : \mathrm{MA} + \mathrm{PB} \xrightarrow{50\cdot \mathrm{MA}\cdot \mathrm{PB}} \mathrm{PB}$<br>$degradeB : \mathrm{MB} + \mathrm{PC} \xrightarrow{50\cdot \mathrm{MB}\cdot \mathrm{PC}} \mathrm{PC}$<br>$degradeC : \mathrm{MC} + \mathrm{PA} \xrightarrow{50\cdot \mathrm{MC}\cdot \mathrm{PA}} \mathrm{PA}$<br>$dissolveA : \mathrm{PA} \xrightarrow{0.01\cdot \mathrm{PA}} \emptyset$<br>$dissolveB : \mathrm{PB} \xrightarrow{0.01\cdot \mathrm{PB}} \emptyset$<br>$dissolveC : \mathrm{PC} \xrightarrow{0.01\cdot \mathrm{PC}} \emptyset$ |

| **Viral Infection (VI)** | |
|---|---|
| Species (4) | DNA, RNA, P, V |
| Initial state | $(1 \times \mathrm{RNA})$ |
| End time | $200s$ |
| Reactions<br>(6) | $d0 : \mathrm{DNA} + \mathrm{P} \xrightarrow{1.125E-5\cdot \mathrm{DNA}\cdot \mathrm{P}} \mathrm{V}$<br>$x : \mathrm{RNA} \xrightarrow{1000\cdot \mathrm{RNA}} \mathrm{RNA} + \mathrm{P}$<br>$t : \mathrm{DNA} \xrightarrow{0.025\cdot \mathrm{DNA}} \mathrm{DNA} + \mathrm{RNA}$<br>$p : \mathrm{RNA} \xrightarrow{1\cdot \mathrm{RNA}} \mathrm{DNA} + \mathrm{RNA}$<br>$d2 : \mathrm{RNA} \xrightarrow{0.25\cdot \mathrm{RNA}} \emptyset$<br>$d5 : \mathrm{P} \xrightarrow{1.9985\cdot \mathrm{P}} \emptyset$ |

behavior. Therefore, they are commonly used to evaluate advanced numerical as well as simulation methods.

The experiment was carried out on a machine with an Intel Core i7-11700K @ 3.60GHz CPU and 8GB of RAM. We use our own competitive implementation of the direct SSA method as a baseline that achieves between $1 \times 10^6$ and $7 \times 10^6$ reactions per second, depending on the model. Segmental simulation has multiple hyper-parameters that impact accuracy and performance. In this work, we choose to evaluate our method with a population-level growth factor of 1.5 and 100 segments per abstract state. For a more detailed discussion of hyper-parameters, see [HČK+22] or Appendix C.

### 7.5.1 Accuracy

For each model, we evaluate the accuracy of simulation techniques in two ways: First, we present a qualitative accuracy evaluation by visually comparing simulations that are plotted in Figure 7.5. Then, we perform a quantitative accuracy evaluation using Table 7.3, which shows how model-specific properties are preserved.

**Table 7.3:** Accuracy evaluation of approximate simulation techniques after $10^4$ simulations via model-specific properties.

| Model | Property | SSA | HYB | SEG | HYBSEG |
|---|---|---|---|---|---|
| PP | oscillation period: mean (s.d.) | 7.1*s* (1.1*s*) | 7.1*s* (1.1*s*) | 6.9*s* (1.4*s*) | 7.0*s* (1.3*s*) |
| | oscillation PRED-peak: mean (s.d.) | 500 (240) | 530 (250) | 420 (200) | 450 (220) |
| | die-out probability: | 70.8% | 86.7% | 42.2% | 57.4% |
| | die-out reason: PRED, PREY | 48%, 52% | 48%, 52% | 50%, 50% | 52%, 48% |
| RP | oscillation period: mean (s.d.) | 2600*s* (240*s*) | 2700*s* (240*s*) | 2500*s* (340*s*) | 2500*s* (320*s*) |
| | oscillation PA-peak: mean (s.d.) | 5.0*e*4 (1.4*e*4) | 5.5*e*4 (1.1*e*4) | 4.5*e*4 (1.5*e*4) | 4.5*e*4 (1.5*e*4) |
| | dominance (at 50000*s*, in %): A, B, C | 23, 40, 37 | 45, 15, 40 | 35, 33, 32 | 36, 30, 33 |
| TS | switches per simulation: | 1.9 (1.3) | 2.0 (1.4) | 1.0 (1.1) | 1.0 (1.1) |
| | P-peak: mean (s.d.) | 1.2*e*4 (1.1*e*3) | 1.2*e*4 (1.1*e*3) | 1.3*e*4 (2.1*e*3) | 1.3*e*4 (2.0*e*3) |
| | dominance (at 50000*s*, in %): A, B | 50, 50 | 49, 51 | 48, 52 | 46, 54 |
| VI | die-out probability: | 20.3% | 19.4% | 19.6% | 20.4% |
| | DNA at 200*s*[a]: mean (s.d.) | 174 (25) | 173 (25) | 170 (25) | 169 (27) |
| | RNA at 200*s*[a]: mean (s.d.) | 17.3 (4.9) | 17.3 (4.9) | 16.9 (5.3) | 16.7 (5.3) |
| | P at 200*s*[a]: mean (s.d.) | 8600 (2400) | 8700 (2400) | 8200 (2600) | 8200 (2700) |
| | V at 200*s*[a]: mean (s.d.) | 2100 (630) | 2100 (640) | 2000 (640) | 2000 (670) |

[a]Simulations that die out are ignored.

**Predator Prey.** All simulations of Figure 7.5 show the expected anti-cyclic oscillation between both species with varying amplitude. The simulation can end early if one of the species dies out. All approximate simulation techniques closely match SSA's mean oscillation frequency of 7.1 seconds and standard derivation of 1.1 seconds. The die-out behavior of the model is notoriously hard to preserve for approximate simulation methods. While they all correctly predict that it is equally likely for either species to die out first, they cannot correctly predict the number of runs that die in the first 200

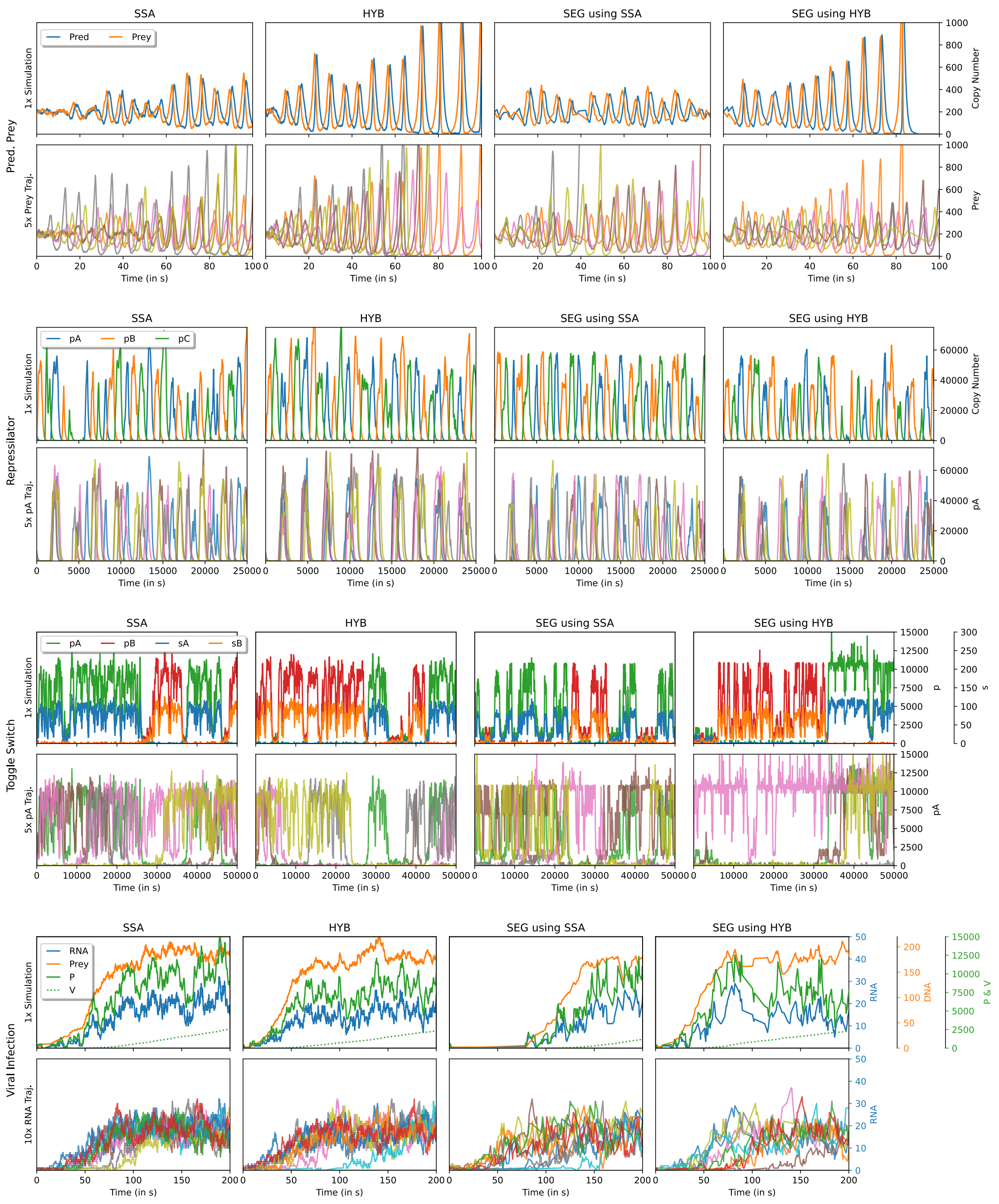


**Figure 7.5:** Visual comparison of simulation approaches: SSA, HYB, SEG, and HYBSEG. For each model, the top row presents a single simulation, emphasizing the relations among different species. The bottom row illustrates the stochasticity with multiple trajectories for a single species.

seconds. SSA tells us that the die-out probability should be 71%, but HYB overestimates with 87%, and SEG (HYBSEG) underestimates with 43% (57%).

**Repressilator.** All approaches produce the expected alternating peaks of the three P-species (see Figure 7.5). In a visual comparison, we find that all produced simulations look very similar and are thus hard to distinguish. In a quantitative comparison, we find that the average oscillation period of 2600*s* is matched by all approximate approaches. However, both SEG and HYBSEG predict slightly more variance in the oscillation frequency. As a result, they also predict that it is equally like for either species to be dominant at $t$=50000*s* while it is actually less likely for A to be the dominant species. On the other hand, we find that HYB underestimates the variance of the PA-peaks as it treats the large copy numbers as deterministic.

**Toggle Switch.** A qualitative comparison of simulation plots shows that all techniques predict the expected switching of periods with A-species dominance and B-species dominance. Visually, a few copy numbers (e.g., 11000 for P-species) are noticeably too dominant in the segmental simulations. This is a result of a strong correlation between the copy number of S- and P-species and the abstraction-based rounding. Similarly, we find that segmental simulations contain, on average, only 1.0 switches compared to the 1.9 switches in SSA simulations. All techniques preserve the average peak height of P-species between switches and correctly predict that both A-species and B-species are equally likely to be dominant at $t$=50000*s*.

**Viral Infection.** We find that all approximate simulation approaches correctly predict the typical behavior of the system: Initially, there is a small chance of 20% that the system dies out. Otherwise, all species grow until they reach a level that they oscillate around (see Figure 7.5). To quantify the accuracy, we compare the RNA distribution at $t$=200*s*: SSA predicts a bi-modal distribution where 20.3% die out and the other cases are distributed normally with a mean of 17.3 and a standard deviation of 4.9. HYB matches the RNA distribution perfectly but with a slightly lower death percentage (19.4%). The segmental simulation approaches match the death percentage better but do have slightly worse accuracy for the RNA distribution.

**Accuracy Conclusion.** Our investigation reveals that both SEG and HYBSEG, being approximate simulation techniques, introduce measurable inaccuracies. These techniques can underestimate the likelihood of unlikely events, such as the switching in TS, and struggle to capture sensitive transient properties, like the die-out behavior in PP. Notably, similar inaccuracies are observed in the hybrid simulation results. This suggests that segmental simulation exhibits accuracy comparable to other approximate simulation techniques. However, we find that segmental simulation successfully preserves key system dynamics, as evidenced by the visual similarity between its simulations and SSA simulations. And, more importantly, it accurately predicts the majority of model-specific quantities.

### 7.5.2 Speedup

Recall that segmental simulation is based on reusing segments generated in previous simulation runs. As such, it becomes faster with each subsequent simulation of the system. This can be observed in Figure 7.6, which shows the speedup we achieve when generating an increasing number of simulations with segmental simulation instead of SSA. Typically, segmental simulation does not speed up the very first simulation as the memory starts empty. However, for some models, segments can already be reused in the first simulation, like in the oscillating TS model, where the speedup for the first SSA-based segmental simulation is already 15x. The speedup grows rapidly during the early simulations, when segments for the most important abstract states are generated, and approaches a limit where segments are reused as much as possible.

**Table 7.4:** Average run-time of one SSA simulation and the relative speedup w.r.t. SSA when computing $10^4$ simulations with other methods.

| Model | SSA | HYB | SEG using … | |
|---|---|---|---|---|
| | | | … SSA | … HYB |
| PP | 0.013$s$ | 1.2x | 46x | 48x |
| RP | 8.4$s$ | 24x | 160x | 500x |
| TS | 21$s$ | 12x | 280x | 310x |
| VI | 0.74$s$ | 68x | 150x | 3300x |

Table 7.4 shows the speedup factors for different simulation methods when generating $10^4$ simulations. On its own, SEG already speeds up the simulation process significantly. The lowest speedup is 46x for the PP model and the highest speedup is achieved for TS with 280x. For models where hybrid simulation is effective, like TS, RP, and VI, we can achieve even larger speedups by combining SEG with our abstraction-based hybrid approach. In VI, the speedups of HYB and SEG on their own are 68x and 150x, respectively, but the combination of both approaches results in a speedup of 3300x.

### 7.5.3 Memory

Segmental simulation, being a memoization approach, comes with the trade-off of increased memory consumption in exchange for speedup. For our benchmarks, we observed relatively modest memory requirements, as indicated in Table 7.5. Notably, by utilizing the segment distribution approximation illustrated in Figure 7.3, it is possible to reduce total memory consumption by approximately 65% to 85%.

**Reusing Segments for Different Simulation Tasks.** In the experiments described so far, the memory of segmental simulation starts empty and is filled over time. The filled memory can be understood as an artifact speeding up future segmental simulations of the same model. This is the case even if future simulations start from a different initial state or end at a different time. Consider the scenario depicted in Figure 7.7 where a

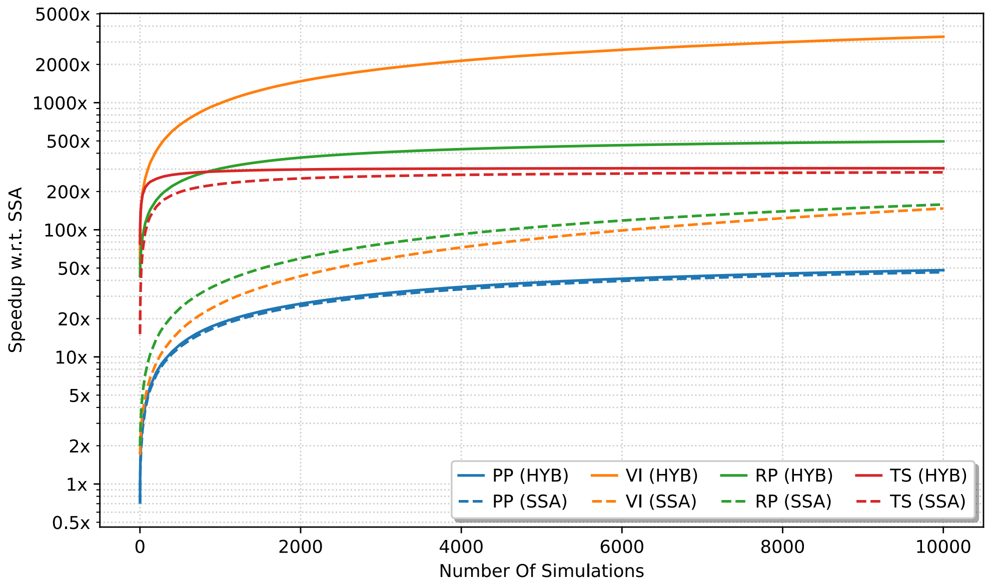


**Figure 7.6:** Speedup achieved by segmental simulation over SSA when generating a given number of simulations.

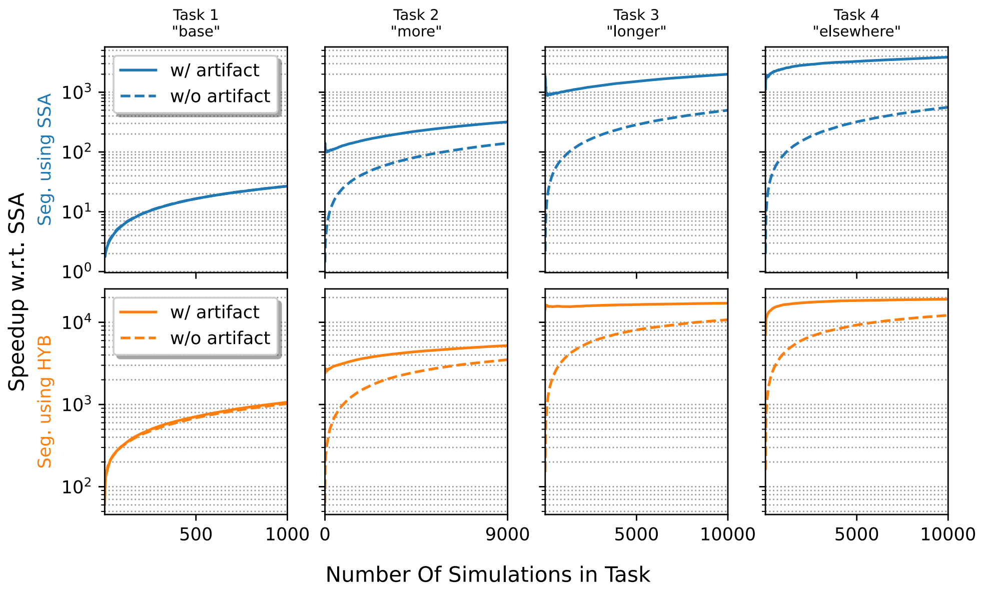


**Figure 7.7:** Speedup per simulation task for a sequence of transient analyses in VI: (1) 1000 simulations, (2) 9000 simulations, (3) 10x longer simulation time, and (4) starting with a different initial state (5 molecules of RNA). The speedup is significantly improved if the memory between tasks is reused (w/ artifact) instead of emptied (w/o artifact).

**Table 7.5:** Memory usage and the number of abstract states/segments in memory after 10.000 segmental simulations: with and without segment distribution approximation.

| Model | abstract states | Segments | | Memory | |
|---|---|---|---|---|---|
| | | total | in approxim. | w/o approxim. | w/ approxim. |
| PP | 300 | 38,000 | 5,700 (15%) | 2.6MB | 390kB |
| RP | 29,000 | 2.4E6 | 6.2E5 (26%) | 230MB | 61MB |
| TS | 10,000 | 7.2E5 | 2.5E5 (35%) | 69MB | 24MB |
| VI | 430 | 42,000 | 10,000 (24%) | 3.5MB | 840kB |

user performs a series of transient analyses for the VI model: First, they compute 1000 simulations (Task 1), then they need 9000 more simulations to estimate the likelihood of a rare event (Task 2), then they run the simulation ten times longer to make sure the system is stable (Task 3), and finally they modify the initial state to analyze its impact on the system (Task 4). If the memory is emptied between tasks (dashed line), we observe the normal speedup increase per task. However, if the memory is reused between tasks (solid line), then the initial speedup and final speedup per task are larger as less time is spent on generating segments.

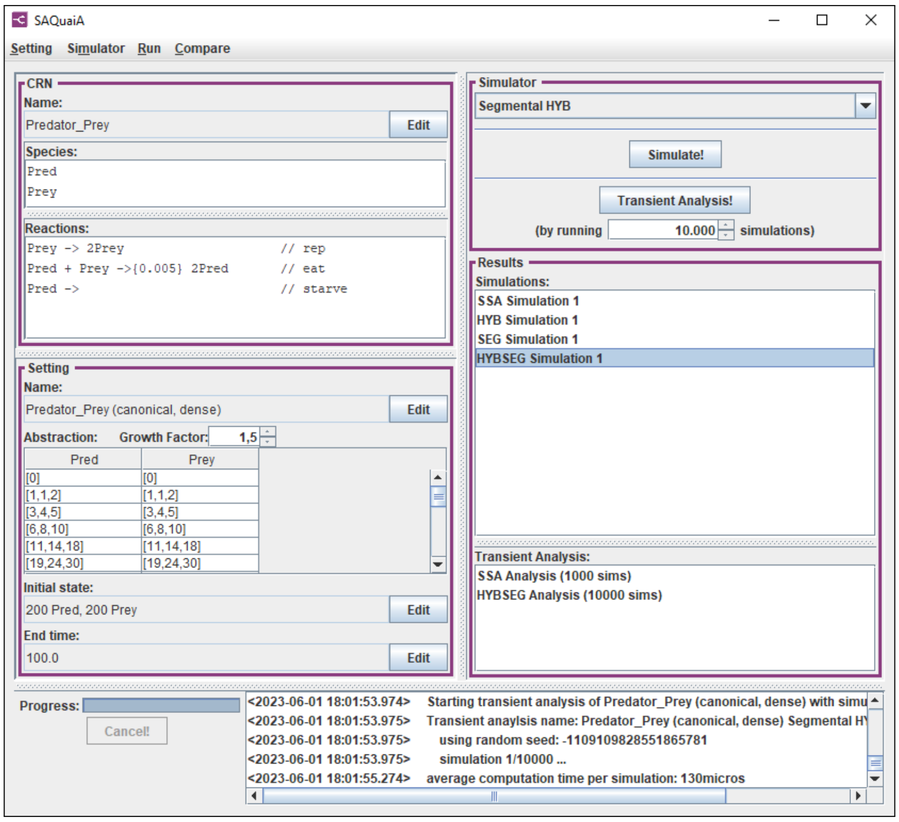
## 7.6 Tool: SAQuaiA

We implemented segmental simulation in the free tool SAQuaiA.[4] In SAQuaiA, users can easily create, simulate, and analyze chemical reaction networks using an intuitive graphical user interface. This enables the rapid application of our segmental simulation approach to novel systems. Furthermore, SAQuaiA offers a range of visualization capabilities that help to interpret results.

**Creating and Modifying Settings.** A *setting* in SAQuaiA consists of a CRN, an initial state, a final time, and a population-level abstraction. Users can create a new setting or modify one of the many provided example settings. The current setting is displayed on the left of the main window (see Figure 7.8), showing the species and reaction as lists as well as the population levels as a table. The abstraction can be changed using the population-growth factor or by adding custom levels.

**Simulators.** SAQuaiA implements six simulation approaches: SSA, our hybrid approach (HYB), segmental simulation using SSA (SEG) and using HYB (HYBSEG) as well as a deterministic simulator (ODE) and a $\tau$-simulation based on our hybrid approach (TAU). The user can quickly change between different approaches (see top right of Figure 7.8), enabling a straightforward comparison between different simulation

[4]SAQuaiA stands for "Simulation & Abstraction-based Quantitative Analyzer". The tool and its predecessor SeQuaiA [ČCK20] are available at https://sequaia.model.in.tum.de.



CRN
Name: Predator_Prey

Species:
```
Pred
Prey
```

Reactions:
```
Prey -> 2Prey                 // rep
Pred + Prey ->{0.005} 2Pred   // eat
Pred ->                       // starve
```

Setting
Name: Predator_Prey (canonical, dense)

Abstraction: Growth Factor: 1,5

| Pred | Prey |
|---|---|
| [0] | [0] |
| [1,1,2] | [1,1,2] |
| [3,4,5] | [3,4,5] |
| [6,8,10] | [6,8,10] |
| [11,14,18] | [11,14,18] |
| [19,24,30] | [19,24,30] |

Initial state: 200 Pred, 200 Prey

End time: 100.0

Simulator: Segmental HYB

(by running 10.000 simulations)

Results
Simulations:
SSA Simulation 1
HYB Simulation 1
SEG Simulation 1
HYBSEG Simulation 1

Transient Analysis:
SSA Analysis (1000 sims)
HYBSEG Analysis (10000 sims)

Progress:

```
<2023-06-01 18:01:53.974>   Starting transient analysis of Predator_Prey (canonical, dense) with simu
<2023-06-01 18:01:53.975>   Transient anaylsis name: Predator_Prey (canonical, dense) Segmental H
<2023-06-01 18:01:53.975>      using random seed: -1109109828551865781
<2023-06-01 18:01:53.975>      simulation 1/10000 ...
<2023-06-01 18:01:55.274>   average computation time per simulation: 130micros
```

**Figure 7.8:** The tool SAQuaiA allows users to simulate and analyze chemical reaction networks.

techniques. Furthermore, the tool allows to modify the hyper-parameters of all approaches directly through the graphical user interface.

**Simulation and Transient Analysis.** There are two options to use the selected simulator for the current setting. The first option is to perform a single simulation. This produces a full evolution of the system over the simulation period. The other option is to perform a transient analysis. This performs a large number of simulations to approximate the transient distribution of the system. For each simulation in a transient analysis, the result contains the final state as well as how long the simulation took. In both use cases, a progress bar and a log displayed at the bottom of the interface provide real-time updates on the computation's progress, keeping the user informed (see Figure 7.8).

**Segmental Simulation for Sequences of Tasks.** SAQuaiA automatically reuses segments from previous analysis tasks to speed up subsequent queries. This makes it possible to speed up sequences of analyses as described in Section 7.5.3. The memory is only reset if the setting or simulator changes.

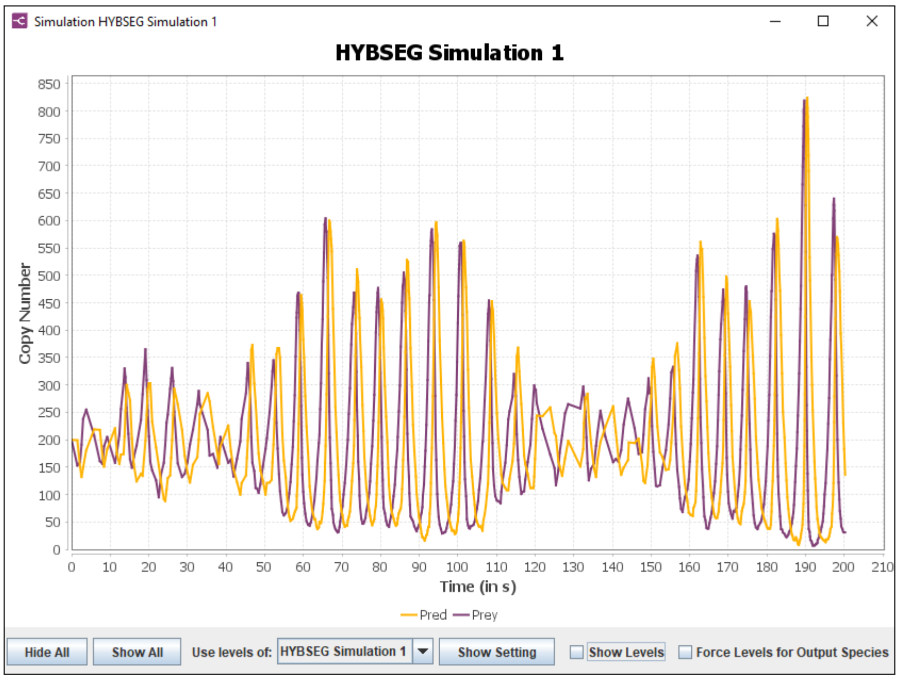


**Figure 7.9:** SAQuaiA: Segmental simulation of the predator-prey system.

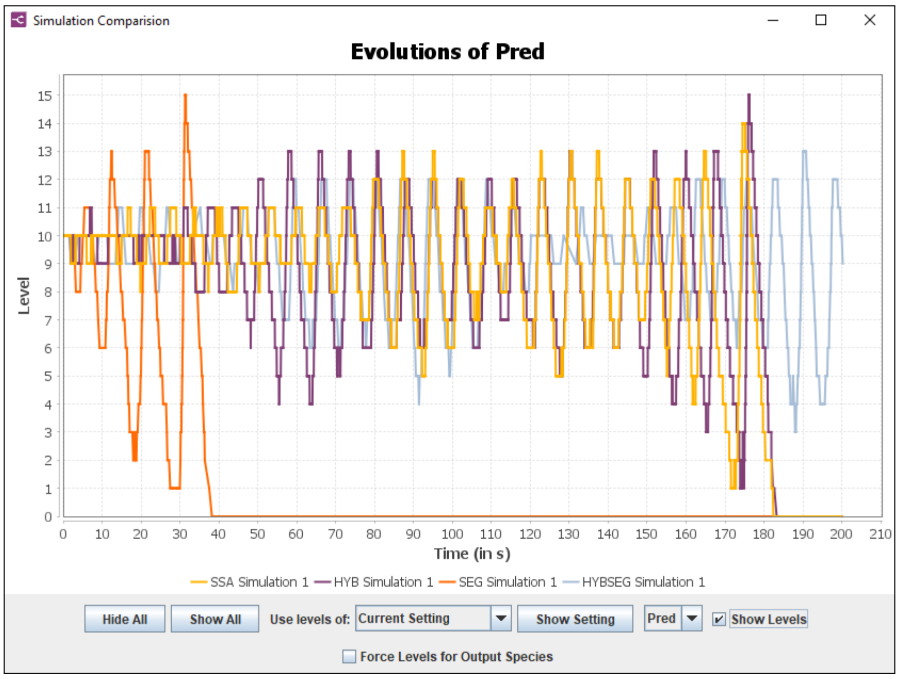


**Figure 7.10:** SAQuaiA: Comparing trajectories of multiple simulations.

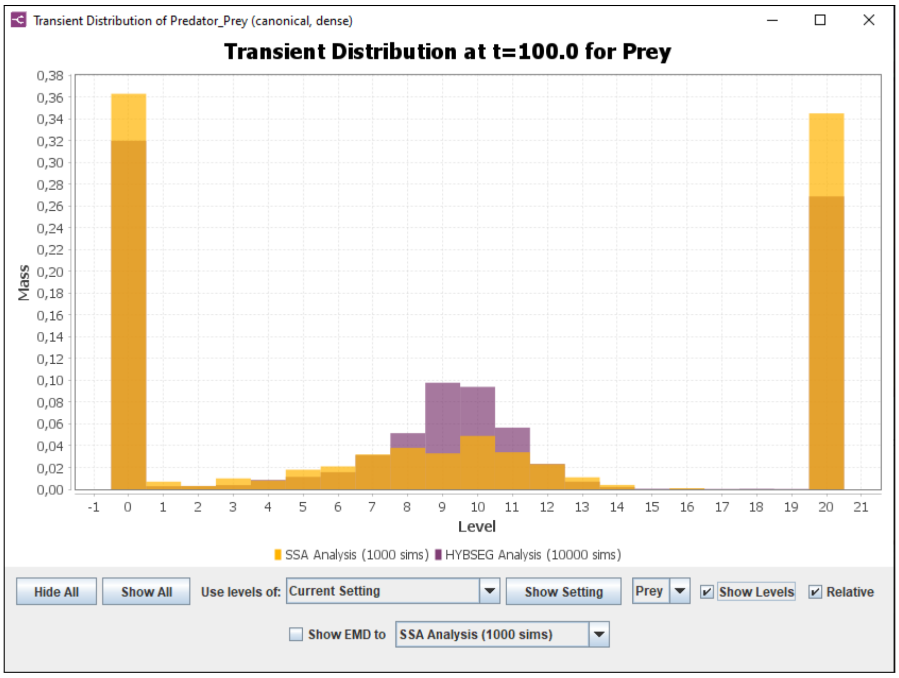


**Figure 7.11:** SAQuaiA: Comparing multiple transient distributions.

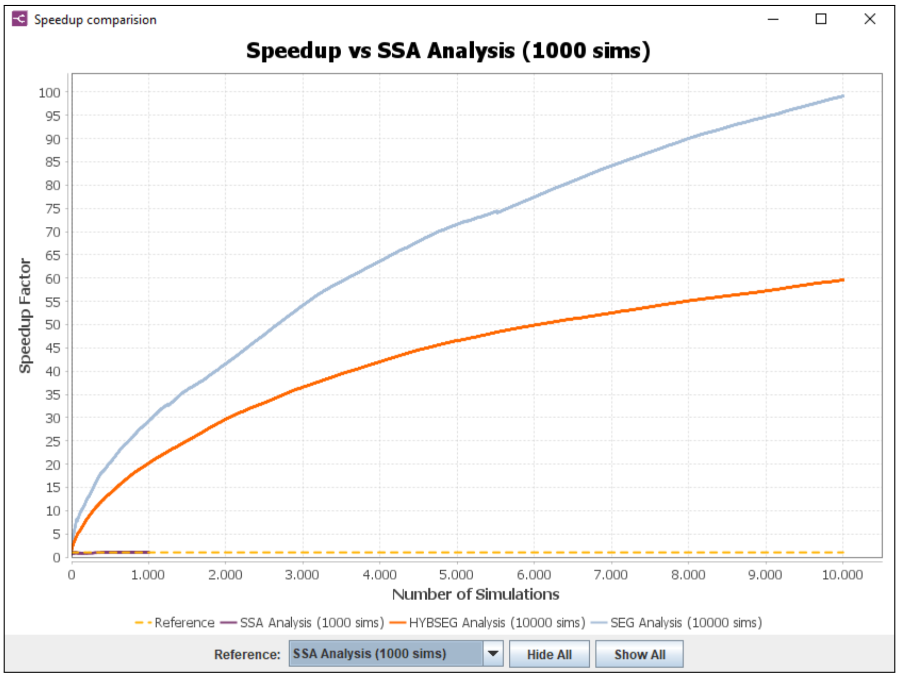


**Figure 7.12:** SAQuaiA: Speed comparison of different simulation techniques.

**Result Visualization.** SAQuaiA can visualize results in multiple different ways:

- For a single simulation, the tool generates a plot displaying individual trajectories for each species in the chemical reaction network (see Figure 7.9).
- Multiple simulations can be compared by plotting their trajectories for a specific species, allowing for a visual comparison (see Figure 7.10).
- Transient analyses are compared by plotting the transient distributions they predict as histograms (see Figure 7.11).
- Comparing the run times of transient analyses makes it possible to visualize the relative efficiency of different simulation approaches (see Figure 7.12).

In all visualizations, it is possible to choose between copy numbers (like in Figure 7.9) or population levels (like in Figure 7.11). Additionally, SAQuaiA enables real-time visualization of simulations and ongoing transient analyses, with plots that update regularly to reflect the evolving data.

## 7.7 Related Work

**Exact Stochastic Simulation.** Gillespie introduced two versions of the stochastic simulation algorithm (SSA) [Gil77]: the *direct method* presented in Section 6.3 and the *first reaction method*. In each simulation step and for each reaction, the first reaction method generates a putative time at which that reaction occurs and then chooses the reaction that occurs first. Both of these methods are exact in the sense that they produce trajectories that are sampled according to the underlying CTMC of the system. Many improvements to SSA have been proposed. The *next reaction method* by Gibson and Bruck [GB00] improves on the first reaction method: Using a dependency graph, it only updates propensities when necessary and it reuses putative times that are ordered in a priority queue to determine the next reaction. The *optimized* version of the *direct method* [CLP04] also uses a dependency graph for updating propensities and orders reactions by their frequency in pre-simulations to improve the random selection of reactions. The *sorting direct method* [MPC+06] builds on the optimized direct method but updates reaction frequencies on-the-fly. *Rejection-based SSA* [TPZ14] maintains a lower and upper bound for the propensities of all reactions. In many cases, this makes it possible to sample the next reaction without calculating propensities but may result in a small number of rejected samples. Stochastic simulation has been further accelerated using parallel computing, for example, on clusters [CLM+05] and GPUs [KD12].

**Approximate Stochastic Simulation.** Next to these exact techniques that apply one reaction at a time, approximate simulation techniques speed up the simulation process by applying multiple reactions at once. One such technique is *$\tau$-leaping* [Gil01] (see Section 6.3.1), which assumes that propensities do not change significantly within the time window $\tau$. This assumption makes it possible to sample the number of occurrences of each reaction using a Poisson distribution. The selection of the parameter $\tau$ was addressed in multiple works such as [GP03], [CGP06], and [LYG+15]. The *Chemical Langevin method* further approximates $\tau$-leaping in cases where we expect many occurrences of all reactions by replacing the Poisson distribution with a Normal distribution [Gil02]. Alternatively, various partitioning schemes have been explored that address the computational challenges posed by fast and slow reactions [CGP04]. One common approach is to approximate fast reactions using a quasi-steady-state assumption [RA03; Gou05]. In *hybrid simulations*, slow and fast sub-networks are separated, allowing to treat certain species as continuous variables and others as discrete [SK05]. The effectiveness and accuracy of these partitioning techniques heavily depend on the appropriate assignment of species. Consequently, several strategies, including adaptive approaches, have been proposed to improve performance and accuracy [GAK15; HGK15]. For surveys comparing multiple exact and approximate simulation techniques, we refer to [Gil07] and [SRP+19].

**Deep Learning.** Recently, the analysis of chemical reaction networks has witnessed the integration of deep learning methodologies to enhance scalability. Cairoli *et al.* introduced a deep learning paradigm for this purpose [CCB21]. In their approach, a generative adversarial network learns from a set of stochastic simulations to generate trajectories that closely resemble the distribution of trajectories in the original CRN. Gupta *et al.* expanded on this concept and developed an estimator that learns statistical properties of the original CRN from simulations [GSK21]. However, these approaches are limited by the computational overhead associated with the learning phase, which typically requires a significant number of simulations of the original CRN that need to be generated with other simulation techniques.

## 7.8 Open Research Questions

The memory requirements of segmental simulation grow exponentially with the number of dimensions. Therefore, it remains to be studied how the generalized segmental simulation scheme of Section 7.3 can be best used to speed up the simulation of larger systems. In these cases, it is important that segmental simulation's memory is adaptive, i.e., that used memory can be repurposed when the memory is full. Otherwise, a change in the approximation of some abstract state's behavior could render large parts of the memory ineffective, for example, because some abstract states are no longer visited.

In addition to SAQuaiA, several other tools exist that support stochastic simulation, including COPASI [HSG+06], Stochkit [SWR+11], iBioSim [MBJ+09], and CERENA [KFR+16]. Integrating segmental simulation into these tools would serve to increase awareness of this new technique and facilitate fair and straightforward comparisons with other approximate simulation approaches. Furthermore, the integration would likely result in even faster simulations, as segmental simulation could leverage the highly optimized versions of the SSA algorithm already present in these tools.

A comprehensive theoretical analysis of the approximation error in segmental simulation would help to understand its suitability for various scenarios. The establishment of a formal error bound, similar to the error bound found in $\tau$-leaping, would provide a robust measure of the algorithm's accuracy, further bolstering confidence in its effectiveness.

The large speedups achieved by segmental simulation make it interesting for parameter estimation. For instance, in systems where a rate constant is unknown but transient behavior is known, it may be possible to predict the parameter's value using a binary search approach, leveraging segmental simulation's ability to rapidly verify transient behavior. A more involved strategy might first run a segmental transient analysis for two extreme values of a parameter and then estimate the behavior of the system for other values by interpolating the segment distribution approximations in each abstract state.

# Bibliography

[AAČ+21] Alessandro Abate, Roman Andriushchenko, Milan Češka, and Marta Kwiatkowska. "Adaptive formal approximations of Markov chains". In: *Performance Evaluation* 148 (2021), p. 102207. DOI: 10.1016/j.peva.2021.102207 (cited on pages 3, 66).

[AAD+04] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. "Computation in Networks of Passively Mobile Finite-State Sensors". In: PODC '04. St. John's, Newfoundland, Canada: Association for Computing Machinery, 2004, pp. 290–299. DOI: 10.1145/1011767.1011810 (cited on pages 1, 3, 8, 47, 48).

[AAD+06] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. "Computation in networks of passively mobile finite-state sensors". In: *Distributed Computing* 18.4 (Mar. 2006), pp. 235–253. DOI: 10.1007/s00446-005-0138-3 (cited on pages 1, 3, 8, 39).

[AAE+07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. "The computational power of population protocols". In: *Distributed Computing* 20.4 (Nov. 2007), pp. 279–304. DOI: 10.1007/s00446-007-0040-2 (cited on pages 2, 3, 11, 46, 47).

[AAE+17] Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. "Time-Space Trade-offs in Population Protocols". In: *Proceedings of the 2017 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2017, pp. 2560–2579. DOI: 10.1137/1.9781611974782.169 (cited on pages 3, 44).

[AAE07] Dana Angluin, James Aspnes, and David Eisenstat. "A Simple Population Protocol for Fast Robust Approximate Majority". In: *Distributed Computing*. Ed. by Andrzej Pelc. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 20–32. DOI: 10.1007/978-3-540-75142-7_5 (cited on page 3).

[AAE08] Dana Angluin, James Aspnes, and David Eisenstat. "Fast computation by population protocols with a leader". In: *Distributed Computing* 21.3 (Sept. 2008), pp. 183–199. DOI: 10.1007/s00446-008-0067-z (cited on pages 3, 13, 51).

[Abd12] Parosh Aziz Abdulla. "Regular model checking". In: *International Journal on Software Tools for Technology Transfer* 14.2 (Apr. 2012), pp. 109–118. DOI: 10.1007/s10009-011-0216-8 (cited on page 43).

[ACJ+96] Parosh Aziz Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. “General Decidability Theorems for Infinite-State Systems”. In: *11th Annual IEEE Symposium on Logic in Computer Science, LICS 1996, New Brunswick, New Jersey, USA, July 27-30, 1996, Proceedings*. IEEE Computer Society, 1996, pp. 313–321. DOI: 10.1109/LICS.1996.561359 (cited on page 31).

[AGV15] Dan Alistarh, Rati Gelashvili, and Milan Vojnović. “Fast and Exact Majority in Population Protocols”. In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC ’15. Donostia-San Sebastián, Spain: Association for Computing Machinery, 2015, pp. 47–56. DOI: 10.1145/2767386.2767429 (cited on pages 3, 16, 38, 39).

[Ang87] Dana Angluin. “Learning regular sets from queries and counterexamples”. In: *Information and Computation* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6 (cited on page 44).

[AR09] James Aspnes and Eric Ruppert. “An Introduction to Population Protocols”. In: *Middleware for Network Eccentric and Mobile Applications*. Ed. by Benoît Garbinato, Hugo Miranda, and Luís Rodrigues. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 97–120. DOI: 10.1007/978-3-540-89707-1_5 (cited on pages 3, 8).

[Ari85] Aristotle. *Complete Works of Aristotle, Volume 2. The Revised Oxford Translation*. Ed. by Jonathan Barnes. Princeton: Princeton University Press, 1985. DOI: 10.1515/9781400835850 (cited on page 1).

[ARZ+15] Benjamin Aminof, Sasha Rubin, Florian Zuleger, and Francesco Spegni. “Liveness of Parameterized Timed Networks”. In: *Automata, Languages, and Programming*. Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 375–387. DOI: 10.1007/978-3-662-47666-6_30 (cited on page 43).

[Asp17] James Aspnes. “Clocked Population Protocols”. In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC ’17. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 431–440. DOI: 10.1145/3087801.3087836 (cited on pages 3, 44).

[Bar53] Maurice S. Bartlett. “Stochastic Processes or the Statistics of Change”. In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 2.1 (1953), pp. 44–64. DOI: 10.2307/2985327 (cited on page 3).

[BBB13] Joffroy Beauquier, Peva Blanchard, and Janna Burman. “Self-stabilizing Leader Election in Population Protocols over Arbitrary Communication Graphs”. In: *Principles of Distributed Systems*. Ed. by Roberto Baldoni, Nicolas Nisse, and Maarten van Steen. Cham: Springer International Publishing, 2013, pp. 38–52. DOI: 10.1007/978-3-319-03850-6_4 (cited on page 3).

[BCG89] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. "Reasoning about Networks with Many Identical Finite State Processes". In: *Information and Computation* 81.1 (1989), pp. 13–31. DOI: 10.1016/0890-5401(89)90026-6 (cited on page 43).

[BEG+20] Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. "Succinct Population Protocols for Presburger Arithmetic". In: *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Ed. by Christophe Paul and Markus Bläser. Vol. 154. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 40:1–40:15. DOI: 10.4230/LIPIcs.STACS.2020.40 (cited on pages 4–6, 48, 50, 53, 158).

[BEH+20] Michael Blondin, Javier Esparza, Martin Helfrich, Antonín Kučera, and Philipp J. Meyer. "Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling". In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 372–397. DOI: 10.1007/978-3-030-53291-8_20 (cited on pages 4–6, 39, 98, 126).

[BEJ+17] Michael Blondin, Javier Esparza, Stefan Jaax, and Philipp J. Meyer. "Towards Efficient Verification of Population Protocols". In: PODC '17. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 423–430. DOI: 10.1145/3087801.3087816 (cited on pages 28, 43).

[BEJ18a] Michael Blondin, Javier Esparza, and Stefan Jaax. "Large Flocks of Small Birds: on the Minimal Size of Population Protocols". In: *35th Symposium on Theoretical Aspects of Computer Science (STACS 2018)*. Ed. by Rolf Niedermeier and Brigitte Vallée. Vol. 96. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 16:1–16:14. DOI: 10.4230/LIPIcs.STACS.2018.16 (cited on pages 3, 38, 39).

[BEJ18b] Michael Blondin, Javier Esparza, and Stefan Jaax. "Peregrine: A Tool for the Analysis of Population Protocols". In: *Computer Aided Verification*. Ed. by Hana Chockler and Georg Weissenbacher. Cham: Springer International Publishing, 2018, pp. 604–611. DOI: 10.1007/978-3-319-96145-3_34 (cited on page 43).

[BEJ19] Michael Blondin, Javier Esparza, and Stefan Jaax. "Expressive Power of Broadcast Consensus Protocols". In: *30th International Conference on Concurrency Theory (CONCUR 2019)*. Vol. 140. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 31:1–31:16. DOI: 10.4230/LIPIcs.CONCUR.2019.31 (cited on pages 3, 44).

[BEK18] Michael Blondin, Javier Esparza, and Antonín Kucera. "Automatic Analysis of Expected Termination Time for Population Protocols". In: *29th International Conference on Concurrency Theory (CONCUR 2018)*. Ed. by Sven Schewe and Lijun Zhang. Vol. 118. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 33:1–33:16. DOI: 10.4230/LIPIcs.CONCUR.2018.33 (cited on pages 13, 36, 38, 39, 43).

[Ber80] Leonard Berman. "The Complexitiy of Logical Theories". In: *Theoretical Computer Science* 11 (1980), pp. 71–77. DOI: 10.1016/0304-3975(80)90037-7 (cited on page 24).

[BHK+20] Petra Berenbrink, David Hammer, Dominik Kaaser, Ulrich Meyer, Manuel Penschuck, and Hung Tran. "Simulating Population Protocols in Sub-Constant Time per Interaction". In: *28th Annual European Symposium on Algorithms (ESA 2020)*. Ed. by Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders. Vol. 173. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 16:1–16:22. DOI: 10.4230/LIPIcs.ESA.2020.16 (cited on page 44).

[BJK+15] Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015. DOI: 10.2200/S00658ED1V01Y201508DCT013 (cited on page 43).

[BJN+00] Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. "Regular Model Checking". In: *Computer Aided Verification*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 403–418. DOI: 10.1007/10722167_31 (cited on page 43).

[Bri19] Robert Brijder. "Computing with chemical reaction networks: a tutorial". In: *Natural Computing* 18.1 (Mar. 2019), pp. 119–137. DOI: 10.1007/s11047-018-9723-9 (cited on page 55).

[BT05] Ahmed Bouajjani and Tayssir Touili. "On Computing Reachability Sets of Process Rewrite Systems". In: *Term Rewriting and Applications*. Ed. by Jürgen Giesl. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 484–499. DOI: 10.1007/978-3-540-32033-3_35 (cited on page 3).

[CBH+09] Vijaysekhar Chellaboina, Sanjay P. Bhat, Wassim M. Haddad, and Dennis S. Bernstein. "Modeling and analysis of mass-action kinetics". In: *IEEE Control Systems Magazine* 29.4 (2009), pp. 60–78. DOI: 10.1109/MCS.2009.932926 (cited on pages 2, 55).

[CCB21] Francesca Cairoli, Ginevra Carbone, and Luca Bortolussi. "Abstraction of Markov Population Dynamics via Generative Adversarial Nets". In: *Computational Methods in Systems Biology*. Ed. by Eugenio Cinquemani and Loïc Paulevé. Cham: Springer International Publishing, 2021, pp. 19–35. DOI: 10.1007/978-3-030-85633-5_2 (cited on page 82).

[ČCK20] Milan Češka, Calvin Chau, and Jan Křetínský. "SeQuaiA: A Scalable Tool for Semi-Quantitative Analysis of Chemical Reaction Networks". In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 653–666. DOI: 10.1007/978-3-030-53288-8_32 (cited on page 77).

[CDF+11] Julien Clement, Carole Delporte-Gallet, Hugues Fauconnier, and Mihaela Sighireanu. "Guidelines for the Verification of Population Protocols". In: *2011 31st International Conference on Distributed Computing Systems*. 2011, pp. 215–224. DOI: 10.1109/ICDCS.2011.36 (cited on pages 13, 38, 39, 42).

[CE21] Philipp Czerner and Javier Esparza. "Lower Bounds on the State Complexity of Population Protocols". In: *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. PODC'21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 45–54. DOI: 10.1145/3465084.3467912 (cited on page 3).

[CFQ+12] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. "Time-varying graphs and dynamic networks". In: *International Journal of Parallel, Emergent and Distributed Systems* 27.5 (2012), pp. 387–408. DOI: 10.1080/17445760.2012.668546 (cited on page 3).

[CGH+21] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. "Decision Power of Weak Asynchronous Models of Distributed Computing". In: PODC'21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 115–125. DOI: 10.1145/3465084.3467918 (cited on page 6).

[CGH+22] Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. "Fast and Succinct Population Protocols for Presburger Arithmetic". In: *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Ed. by James Aspnes and Othon Michail. Vol. 221. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 11:1–11:17. DOI: 10.4230/LIPIcs.SAND.2022.11 (cited on pages 4–6, 49–53, 174).

[CGP04] Yang Cao, Daniel T. Gillespie, and Linda R. Petzold. "The slow-scale stochastic simulation algorithm". In: *The Journal of Chemical Physics* 122.1 (Dec. 2004). DOI: 10.1063/1.1824902 (cited on page 82).

[CGP06] Yang Cao, Daniel T Gillespie, and Linda R Petzold. “Efficient step size selection for the tau-leaping simulation method”. In: *The Journal of chemical physics* 124.4 (2006), p. 044109. DOI: 10.1063/1.2159468 (cited on pages 69, 70, 82).

[CHL+17] Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Rümmer. “Learning to prove safety over parameterised concurrent systems”. In: *2017 Formal Methods in Computer Aided Design (FMCAD)*. 2017, pp. 76–83. DOI: 10.23919/FMCAD.2017.8102244 (cited on page 44).

[ČK19] Milan Češka and Jan Křetínský. “Semi-quantitative Abstraction and Analysis of Chemical Reaction Networks”. In: *Computer Aided Verification*. Ed. by Isil Dillig and Serdar Tasiran. Cham: Springer International Publishing, 2019, pp. 475–496. DOI: 10.1007/978-3-030-25540-4_28 (cited on page 3).

[CLM+05] Vipin Chaudhary, Feng Liu, Vijay Matta, and Laurence T. Yang. “Parallel Implementations of Local Sequence Alignment: Hardware and Software”. In: *Parallel Computing for Bioinformatics and Computational Biology*. John Wiley & Sons, Ltd, 2005. Chap. 10, pp. 233–264. DOI: https://doi.org/10.1002/0471756504.ch10 (cited on page 81).

[CLP04] Yang Cao, Hong Li, and Linda Petzold. “Efficient formulation of the stochastic simulation algorithm for chemically reacting systems”. In: *The Journal of Chemical Physics* 121.9 (Aug. 2004), pp. 4059–4067. DOI: 10.1063/1.1778376 (cited on page 81).

[CMS10] Ioannis Chatzigiannakis, Othon Michail, and Paul G. Spirakis. “Algorithmic Verification of Population Protocols”. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by Shlomi Dolev, Jorge Cobb, Michael Fischer, and Moti Yung. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 221–235. DOI: 10.1007/978-3-642-16023-3_19 (cited on page 42).

[CO22] Wojciech Czerwiński and Łukasz Orlikowski. “Reachability in Vector Addition Systems is Ackermann-complete”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 1229–1240. DOI: 10.1109/FOCS52979.2021.00120 (cited on pages 16, 25).

[CTT+04] Edmund Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. “Verification by Network Decomposition”. In: *CONCUR 2004 - Concurrency Theory*. Ed. by Philippa Gardner and Nobuko Yoshida. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 276–291. DOI: 10.1007/978-3-540-28644-8_18 (cited on page 43).

[DA94] René David and Hassane Alla. “Petri nets for modeling of dynamic systems: A survey”. In: *Automatica* 30.2 (1994), pp. 175–202. DOI: 10.1016/0005-1098(94)90024-8 (cited on page 3).

[DE95] Jorg Desel and Javier Esparza. *Free choice Petri nets*. 40. Cambridge university press, 1995. DOI: 10.1017/CBO9780511526558 (cited on page 30).

[DF01] Zoë Diamadi and Michael J. Fischer. “A simple game for the study of trust in distributed systems”. In: *Wuhan University Journal of Natural Sciences* 6.1 (Mar. 2001), pp. 72–82. DOI: 10.1007/BF03160228 (cited on page 3).

[DFI+19] Giuseppe A. Di Luna, Paola Flocchini, Taisuke Izumi, Tomoko Izumi, Nicola Santoro, and Giovanni Viglietta. “Population protocols with faulty interactions: The impact of a leader”. In: *Theoretical Computer Science* 754 (2019). Algorithms and Complexity, pp. 35–49. DOI: 10.1016/j.tcs.2018.09.005 (cited on page 3).

[DHK+20] Loris D’Antoni, Martin Helfrich, Jan Kretinsky, Emanuel Ramneantu, and Maximilian Weininger. “Automata Tutor v3”. In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 3–14. DOI: 10.1007/978-3-030-53291-8_1 (cited on page 6).

[DM09] Yuxin Deng and Jean-François Monin. “Verifying Self-stabilizing Population Protocols with Coq”. In: *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. 2009, pp. 201–208. DOI: 10.1109/TASE.2009.9 (cited on page 43).

[Doo45] J. L. Doob. “Markoff Chains–Denumerable Case”. In: *Transactions of the American Mathematical Society* 58.3 (1945), pp. 455–473. DOI: 10.2307/1990339 (cited on page 3).

[DS18] David Doty and David Soloveichik. “Stable leader election in population protocols requires linear time”. In: *Distributed Computing* 31.4 (Aug. 2018), pp. 257–271. DOI: 10.1007/s00446-016-0281-z (cited on page 3).

[DS21] David Doty and Eric Severson. “Ppsim: A Software Package for Efficiently Simulating and Visualizing Population Protocols”. In: *Computational Methods in Systems Biology*. Ed. by Eugenio Cinquemani and Loïc Paulevé. Cham: Springer International Publishing, 2021, pp. 245–253. DOI: 10.1007/978-3-030-85633-5_16 (cited on page 44).

[EGL+17] Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. “Verification of population protocols”. In: *Acta Informatica* 54.2 (Mar. 2017), pp. 191–215. DOI: 10.1007/s00236-016-0272-3 (cited on pages 3, 16, 23–25, 45).

[EGM+18] Javier Esparza, Pierre Ganty, Rupak Majumdar, and Chana Weil-Kennedy. “Verification of Immediate Observation Population Protocols”. In: *29th International Conference on Concurrency Theory (CONCUR 2018)*. Ed. by Sven Schewe and Lijun Zhang. Vol. 118. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 31:1–31:16. DOI: 10.4230/LIPIcs.CONCUR.2018.31 (cited on page 3).

[EHJ+20] Javier Esparza, Martin Helfrich, Stefan Jaax, and Philipp J. Meyer. “Peregrine 2.0: Explaining Correctness of Population Protocols Through Stage Graphs”. In: *Automated Technology for Verification and Analysis*. Ed. by Dang Van Hung and Oleg Sokolsky. Cham: Springer International Publishing, 2020, pp. 550–556. DOI: 10.1007/978-3-030-59152-6_32 (cited on pages 4–6, 126).

[ELM+14] Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp Meyer, and Filip Niksic. “An SMT-Based Approach to Coverability Analysis”. In: *Computer Aided Verification*. Ed. by Armin Biere and Roderick Bloem. Cham: Springer International Publishing, 2014, pp. 603–619. DOI: 10.1007/978-3-319-08867-9_40 (cited on page 28).

[EN03] E. Allen Emerson and Kedar S. Namjoshi. “On Reasoning About Rings”. In: *International Journal of Foundations of Computer Science* 14.04 (2003), pp. 527–549. DOI: 10.1142/S0129054103001881 (cited on page 43).

[ES69] Samuel Eilenberg and M.P Schützenberger. “Rational sets in commutative monoids”. In: *Journal of Algebra* 13.2 (1969), pp. 173–191. DOI: 10.1016/0021-8693(69)90070-2 (cited on page 23).

[FS01] Alain Finkel and Philippe Schnoebelen. “Well-structured transition systems everywhere!” In: *Theoretical Computer Science* 256.1-2 (2001), pp. 63–92. DOI: 10.1016/S0304-3975(00)00102-X (cited on page 31).

[GAK15] Arnab Ganguly, Derya Altintan, and Heinz Koeppl. “Jump-Diffusion Approximation of Stochastic Reaction Dynamics: Error Bounds and Algorithms”. In: *Multiscale Modeling & Simulation* 13.4 (2015), pp. 1390–1419. DOI: 10.1137/140983471 (cited on page 82).

[GB00] Michael A. Gibson and Jehoshua Bruck. “Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels”. In: *The Journal of Physical Chemistry A* 104.9 (Mar. 2000), pp. 1876–1889. DOI: 10.1021/jp993732q (cited on page 81).

[Gil01] Daniel T. Gillespie. “Approximate accelerated stochastic simulation of chemically reacting systems”. In: *The Journal of Chemical Physics* 115.4 (July 2001), pp. 1716–1733. DOI: 10.1063/1.1378322 (cited on pages 59, 81).

[Gil02] Daniel T. Gillespie. “The Chemical Langevin and Fokker - Planck Equations for the Reversible Isomerization Reaction”. In: *The Journal of Physical Chemistry A* 106.20 (May 2002), pp. 5063–5071. DOI: 10.1021/jp0128832 (cited on page 82).

[Gil07] Daniel T. Gillespie. “Stochastic Simulation of Chemical Kinetics”. In: *Annual Review of Physical Chemistry* 58.1 (2007), pp. 35–55. DOI: 10.1146/annurev.physchem.58.032806.104637 (cited on page 82).

[Gil77] Daniel T. Gillespie. “Exact stochastic simulation of coupled chemical reactions”. In: *The Journal of Physical Chemistry* 81.25 (Dec. 1977), pp. 2340–2361. DOI: 10.1021/j100540a008 (cited on pages 3, 55, 59, 70, 81).

[Gil92] Daniel T. Gillespie. "A rigorous derivation of the chemical master equation". In: *Physica A: Statistical Mechanics and its Applications* 188.1 (1992), pp. 404–425. DOI: 10.1016/0378-4371(92)90283-V (cited on pages 3, 58).

[GM84] Donald Gross and Douglas R. Miller. "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes". In: *Operations Research* 32.2 (1984), pp. 343–361. DOI: 10.1287/opre.32.2.343 (cited on page 3).

[Gou05] John Goutsias. "Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems". In: *The Journal of Chemical Physics* 122.18 (May 2005). DOI: 10.1063/1.1889434 (cited on page 82).

[GP03] Daniel T. Gillespie and Linda R. Petzold. "Improved leap-size selection for accelerated stochastic simulation". In: *The Journal of Chemical Physics* 119.16 (Oct. 2003), pp. 8229–8234. DOI: 10.1063/1.1613254 (cited on page 82).

[GR07] Rachid Guerraoui and Eric Ruppert. *Even small birds are unique: Population protocols with identifiers*. Tech. rep. 2007 (cited on page 3).

[Gra77] W.K. Grassmann. "Transient solutions in markovian queueing systems". In: *Computers & Operations Research* 4.1 (1977), pp. 47–53. DOI: 10.1016/0305-0548(77)90007-7 (cited on page 3).

[GS20] Leszek Gasieniec and Grzegorz Stachowiak. "Enhanced Phase Clocks, Population Protocols, and Fast Space Optimal Leader Election". In: *J. ACM* 68.1 (Nov. 2020). DOI: 10.1145/3424659 (cited on pages 3, 44).

[GSK21] Ankit Gupta, Christoph Schwab, and Mustafa Khammash. "DeepCME: A deep learning framework for computing solution statistics of the chemical master equation". In: *PLOS Computational Biology* 17.12 (Dec. 2021), pp. 1–23. DOI: 10.1371/journal.pcbi.1009623 (cited on page 82).

[Haa18] Christoph Haase. "A Survival Guide to Presburger Arithmetic". In: *ACM SIGLOG News* 5.3 (July 2018), pp. 67–82. DOI: 10.1145/3242953.3242964 (cited on page 7).

[HČK+22] Martin Helfrich, Milan Češka, Jan Křetínský, and Štefan Martiček. "Abstraction-Based Segmental Simulation of Chemical Reaction Networks". In: *Computational Methods in Systems Biology*. Ed. by Ion Petre and Andrei Păun. Cham: Springer International Publishing, 2022, pp. 41–60. DOI: 10.1007/978-3-031-15034-0_3 (cited on pages 4–6, 72, 135).

[HGK15] Benjamin Hepp, Ankit Gupta, and Mustafa Khammash. "Adaptive hybrid simulations for multiscale stochastic reaction networks". In: *The Journal of chemical physics* 142.3 (2015), p. 034118. DOI: 10.1063/1.4905196 (cited on pages 62, 69, 70, 82).

[HSG+06] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jürgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. "COPASI—a COmplex PAthway SImulator". In: *Bioinformatics* 22.24 (Oct. 2006), pp. 3067–3074. DOI: 10.1093/bioinformatics/btl485 (cited on page 82).

[IGE00] Chalermek Intanagonwiwat, Ramesh Govindan, and Deborah Estrin. "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks". In: MobiCom '00. Boston, Massachusetts, USA: Association for Computing Machinery, 2000, pp. 56–67. DOI: 10.1145/345910.345920 (cited on page 3).

[JLE77] Neil D. Jones, Lawrence H. Landweber, and Y. Edmund Lien. "Complexity of some problems in Petri nets". In: *Theoretical Computer Science* 4.3 (1977), pp. 277–299. DOI: 10.1016/0304-3975(77)90014-7 (cited on page 32).

[KD12] Ivan Komarov and Roshan M. D'Souza. "Accelerating the Gillespie Exact Stochastic Simulation Algorithm Using Hybrid Parallel Execution on Graphics Processing Units". In: *PLOS ONE* 7.11 (Nov. 2012), pp. 1–9. DOI: 10.1371/journal.pone.0046693 (cited on page 81).

[Ken50] David G. Kendall. "An Artificial Realization of a Simple "Birth-And-Death" Process". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 12.1 (1950), pp. 116–119. DOI: 10.1111/j.2517-6161.1950.tb00048.x (cited on page 3).

[KFR+16] Atefeh Kazeroonian, Fabian Fröhlich, Andreas Raue, Fabian J Theis, and Jan Hasenauer. "CERENA: ChEmical REaction Network Analyzer—a toolbox for the simulation and analysis of stochastic chemical kinetics". In: *PloS one* 11.1 (2016), e0146732. DOI: 10.1371/journal.pone.0146732 (cited on page 82).

[KKW10] Alexander Kaiser, Daniel Kroening, and Thomas Wahl. "Dynamic Cutoff Detection in Parameterized Concurrent Programs". In: *Computer Aided Verification*. Ed. by Tayssir Touili, Byron Cook, and Paul Jackson. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 645–659. DOI: 10.1007/978-3-642-14295-6_55 (cited on page 43).

[KM69] Richard M. Karp and Raymond E. Miller. "Parallel program schemata". In: *Journal of Computer and System Sciences* 3.2 (1969), pp. 147–195. DOI: https://doi.org/10.1016/S0022-0000(69)80011-5 (cited on page 3).

[Kol31] A. Kolmogoroff. "Über die analytischen Methoden in der Wahrscheinlichkeitsrechnung". In: *Mathematische Annalen* 104.1 (Dec. 1931), pp. 415–458. DOI: 10.1007/BF01457949 (cited on page 3).

[Ler12] Jérôme Leroux. "Vector Addition Systems Reachability Problem (A Simpler Solution)". In: *Turing-100. The Alan Turing Centenary*. Ed. by Andrei Voronkov. Vol. 10. EPiC Series in Computing. EasyChair, 2012, pp. 214–228. DOI: 10.29007/bnx2 (cited on page 23).

[Ler22] Jérôme Leroux. "The Reachability Problem for Petri Nets is Not Primitive Recursive". In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2022, pp. 1241–1252. DOI: 10.1109/FOCS52979.2021.00121 (cited on pages 16, 25).

[LLM+17] Ondřej Lengál, Anthony W. Lin, Rupak Majumdar, and Philipp Rümmer. "Fair Termination for Parameterized Probabilistic Concurrent Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by Axel Legay and Tiziana Margaria. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 499–517. DOI: 10.1007/978-3-662-54577-5_29 (cited on page 44).

[LMV22] M Lazarova, S Markov, and A Vassilev. "Dynamical systems induced by reaction networks with application to epidemiological outbreaks". In: *AIP Conference Proceedings*. Vol. 2522. 1. AIP Publishing LLC. 2022, p. 020001. DOI: 10.1063/5.0100921 (cited on pages 2, 55).

[LR16] Anthony W. Lin and Philipp Rümmer. "Liveness of Randomised Parameterised Systems under Arbitrary Schedulers". In: *Computer Aided Verification*. Ed. by Swarat Chaudhuri and Azadeh Farzan. Cham: Springer International Publishing, 2016, pp. 112–133. DOI: 10.1007/978-3-319-41540-6_7 (cited on page 44).

[LYG+15] C. Lester, C. A. Yates, M. B. Giles, and R. E. Baker. "An adaptive multilevel simulation algorithm for stochastic biological systems". In: *The Journal of Chemical Physics* 142.2 (Jan. 2015). DOI: 10.1063/1.4904980 (cited on page 82).

[MB08] Leonardo de Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. R. Ramakrishnan and Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24 (cited on page 38).

[MBJ+09] Chris J. Myers, Nathan Barker, Kevin Jones, Hiroyuki Kuwahara, Curtis Madsen, and Nam-Phuong D. Nguyen. "iBioSim: a tool for the analysis and design of genetic circuits". In: *Bioinformatics* 25.21 (July 2009), pp. 2848–2849. DOI: 10.1093/bioinformatics/btp457 (cited on page 82).

[MCS11] Othon Michail, Ioannis Chatzigiannakis, and Paul G. Spirakis. "Mediated population protocols". In: *Theoretical Computer Science* 412.22 (2011), pp. 2434–2450. DOI: 10.1016/j.tcs.2011.02.003 (cited on page 3).

[MK06] Brian Munsky and Mustafa Khammash. "The finite state projection algorithm for the solution of the chemical master equation". In: *The Journal of Chemical Physics* 124.4 (Jan. 2006). DOI: 10.1063/1.2145882 (cited on page 3).

[MMR+12] Curtis Madsen, Chris J. Myers, Nicholas Roehner, Chris Winstead, and Zhen Zhang. "Utilizing stochastic model checking to analyze genetic circuits". In: *2012 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*. 2012, pp. 379–386. DOI: 10.1109/CIBCB.2012.6217255 (cited on page 66).

[MPC+06] James M. McCollum, Gregory D. Peterson, Chris D. Cox, Michael L. Simpson, and Nagiza F. Samatova. "The sorting direct method for stochastic simulation of biochemical systems with varying reaction execution behavior". In: *Computational Biology and Chemistry* 30.1 (2006), pp. 39–49. DOI: https://doi.org/10.1016/j.compbiolchem.2005.10.007 (cited on page 81).

[MS15a] Othon Michail and Paul G. Spirakis. "Terminating population protocols via some minimal global knowledge assumptions". In: *Journal of Parallel and Distributed Computing* 81-82 (2015), pp. 1–10. DOI: 10.1016/j.jpdc.2015.02.005 (cited on page 3).

[MS15b] Othon Michail and Paul G. Spirakis. "Terminating population protocols via some minimal global knowledge assumptions". In: *Journal of Parallel and Distributed Computing* 81-82 (2015), pp. 1–10. DOI: https://doi.org/10.1016/j.jpdc.2015.02.005 (cited on page 44).

[PLD08] Jun Pang, Zhengqin Luo, and Yuxin Deng. "On automatic verification of self-stabilizing population protocols". In: *Frontiers of Computer Science in China* 2.4 (Dec. 2008), pp. 357–367. DOI: 10.1007/s11704-008-0040-9 (cited on page 42).

[Pre29] Mojżesz Presburger. "Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt". In: *Comptes Rendus du I$^{er}$ Congrès des mathématiciens des pays slaves* (1929), pp. 192–201 (cited on page 24).

[RA03] Christopher V. Rao and Adam P. Arkin. "Stochastic chemical kinetics and the quasi-steady-state assumption: Application to the Gillespie algorithm". In: *The Journal of Chemical Physics* 118.11 (Feb. 2003), pp. 4999–5010. DOI: 10.1063/1.1545446 (cited on page 82).

[SK05] Howard Salis and Yiannis Kaznessis. "Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions". In: *The Journal of Chemical Physics* 122.5 (Jan. 2005). DOI: 10.1063/1.1835951 (cited on page 82).

[SLD+09] Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. "PAT: Towards Flexible Verification under Fairness". In: *Computer Aided Verification*. Ed. by Ahmed Bouajjani and Oded Maler. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 709–714. DOI: 10.1007/978-3-642-02658-4_59 (cited on pages 13, 42).

[SRP+19] Giulia Simoni, Federico Reali, Corrado Priami, and Luca Marchetti. "Stochastic simulation algorithms for computational systems biology: Exact, approximate, and hybrid methods". In: *WIREs Systems Biology and Medicine* 11.6 (2019), e1459. DOI: 10.1002/wsbm.1459 (cited on page 82).

[SSW10] David Soloveichik, Georg Seelig, and Erik Winfree. "DNA as a universal substrate for chemical kinetics". In: *Proceedings of the National Academy of Sciences of the United States of America* 107.12 (2010), pp. 5393–5398. DOI: 10.1073/pnas.0909380107 (cited on pages 2, 55).

[SWR+11] Kevin R. Sanft, Sheng Wu, Min Roh, Jin Fu, Rone Kwei Lim, and Linda R. Petzold. "StochKit2: software for discrete stochastic simulation of biochemical systems with events". In: *Bioinformatics* 27.17 (July 2011), pp. 2457–2458. DOI: 10.1093/bioinformatics/btr401 (cited on page 82).

[SYS+02] R Srivastava, L You, J Summers, and J Yin. "Stochastic vs. deterministic modeling of intracellular viral kinetics". In: *Journal of Theoretical Biology* 218.3 (2002), pp. 309–321. DOI: 10.1006/jtbi.2002.3078 (cited on page 70).

[TPZ14] Vo Hong Thanh, Corrado Priami, and Roberto Zunino. "Efficient rejection-based simulation of biochemical reactions with stochastic noise and delays". In: *The Journal of Chemical Physics* 141.13 (Oct. 2014). DOI: 10.1063/1.4896985 (cited on page 81).

[Ven80] John Venn. "I. On the diagrammatic and mechanical representation of propositions and reasonings". In: *The London, Edinburgh, and Dublin philosophical magazine and journal of science* 10.59 (1880), pp. 1–18. DOI: 10.1080/14786448008626877 (cited on page 19).

[Wol18] Karsten Wolf. "Petri Net Model Checking with LoLA 2". In: *Application and Theory of Petri Nets and Concurrency*. Ed. by Victor Khomenko and Olivier H. Roux. Cham: Springer International Publishing, 2018, pp. 351–362. DOI: 10.1007/978-3-319-91268-4_18 (cited on page 42).

[ZWC09] Jingwei Zhang, Layne T. Watson, and Yang Cao. "Adaptive aggregation method for the Chemical Master Equation". In: *International Journal of Computational Biology and Drug Design* 2.2 (2009), pp. 134–148. DOI: 10.1504/IJCBDD.2009.028825 (cited on page 3).

# Part I

# First Author Publications

# A Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling

This chapter has been published as a **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, Martin Helfrich, Antonín Kučera, and Philipp J. Meyer. “Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling”. In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 372–397. DOI: 10.1007/978-3-030-53291-8_20

**Summary.** We present a sound and complete method for verifying population protocols using Presburger stage graphs. Presburger stage graphs formally describe the computation of population protocols as a series of stages that correspond to irreversible changes in the system state. As such, they can act as a witness for liveness properties like correctness that can be checked independently. Further, we show that if a population protocol is correct, then there is a Presburger stage graph that proves this. Although this yields an algorithm for verifying arbitrary population protocols, it is not very efficient due to the high theoretical complexity of the verification problem. Thus, we introduce an incomplete but efficient procedure that constructs Presburger stage graphs that can verify many population protocols from the literature. We evaluate our approach on a large number of population protocols and show that it can verify instances with many states and transitions.

**Contributions of thesis author.** The author played a pivotal role in the composition and revision of the manuscript. They actively participated in joint discussions and contributed significantly to the development of the theoretical results presented in the paper. Noteworthy individual contributions include the creation of an efficient verification algorithm, extensive exploration for the necessary heuristics, providing a comprehensive algorithm description, as well as implementing and evaluating the approach on a wide range of benchmarks.

# Checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling

Michael Blondin[1], Javier Esparza[2], Martin Helfrich[2], Antonín Kučera[3], and Philipp J. Meyer[2](✉)

[1] Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca
[2] Technical University of Munich, Munich, Germany
{esparza,helfrich,meyerphi}@in.tum.de
[3] Masaryk University, Brno, Czechia
tony@fi.muni.cz

**Abstract.** We present a sound and complete method for the verification of qualitative liveness properties of replicated systems under stochastic scheduling. These are systems consisting of a finite-state program, executed by an unknown number of indistinguishable agents, where the next agent to make a move is determined by the result of a random experiment. We show that if a property of such a system holds, then there is always a witness in the shape of a *Presburger stage graph*: a finite graph whose nodes are Presburger-definable sets of configurations. Due to the high complexity of the verification problem (non-elementary), we introduce an incomplete procedure for the construction of Presburger stage graphs, and implement it on top of an SMT solver. The procedure makes extensive use of the theory of well-quasi-orders, and of the structural theory of Petri nets and vector addition systems. We apply our results to a set of benchmarks, in particular to a large collection of population protocols, a model of distributed computation extensively studied by the distributed computing community.

**Keywords:** Parameterized verification · Liveness · Stochastic systems

## 1 Introduction

Replicated systems consist of a fully symmetric finite-state program executed by an unknown number of indistinguishable agents, communicating by rendez-vous

Michael Blondin is supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Fonds de recherche du Québec – Nature et technologies (FRQNT). Javier Esparza, Martin Helfrich and Philipp J. Meyer have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS). Antonín Kučera is supported by the Czech Science Foundation, grant No. 18-11193S.

or via shared variables [14,16,41,46]. Examples include distributed protocols and multithreaded programs, or abstractions thereof. The communication graph of replicated systems is a clique. They are a special class of *parameterized systems*, i.e., infinite families of systems that admit a finite description in some suitable modeling language. In the case of replicated systems, the (only) parameter is the number of agents executing the program.

Verifying a replicated system amounts to proving that an infinite family of systems satisfies a given property. This is already a formidable challenge, made even harder by the fact that we want to verify liveness (more difficult than safety) against stochastic schedulers. Loosely speaking, stochastic schedulers select the set of agents that should execute the next action as the result of a random experiment. Stochastic scheduling often appears in distributed protocols, and in particular also in population protocols—a model much studied in distributed computing with applications in computational biology[1]—that supplies many of our case studies [9,58]. Under stochastic scheduling, the semantics of a replicated system is an infinite family of finite-state Markov chains. In this work, we study *qualitative* liveness properties, stating that the infinite runs starting at configurations of the system satisfying a precondition almost surely reach and stay in configurations satisfying a postcondition. In this case, whether the property holds or not depends only on the topology of the Markov chains, and not on the concrete probabilities.

We introduce a formal model of replicated systems, based on multiset rewriting, where processes can communicate by shared variables or multiway synchronization. We present a sound and complete verification method called *Presburger stage graphs*. A Presburger stage graphs is a directed acyclic graphs with Presburger formulas as nodes. A formula represents a possibly infinite inductive set of configurations, i.e., a set of configurations closed under reachability. A node $\mathcal{S}$ (which we identify with the set of configurations it represents) has the following property: A run starting at any configuration of $\mathcal{S}$ almost surely reaches some configuration of some successor $\mathcal{S}'$ of $\mathcal{S}$, and, since $\mathcal{S}'$ is inductive, get trapped in $\mathcal{S}'$. A stage graph labels the node $\mathcal{S}$ with a witness of this property in the form of a *Presburger certificate*, a sort of ranking function expressible in Presburger arithmetic. The completeness of the technique, i.e., the fact that for every property of the replicated system that holds there exists a stage graph proving it, follows from deep results of the theory of vector addition systems (VASs) [52–54].

Unfortunately, the theory of VASs also shows that, while the verification problems we consider are decidable, they have non-elementary computational complexity [33]. As a consequence, verification techniques that systematically explore the space of possible stage graphs for a given property are bound to be very inefficient. For this reason, we design an incomplete but efficient algorithm for the computation of stage graphs. Inspired by theoretical results, the algorithm combines a solver for linear constraints with some elements of the theory of well-structured systems [2,39]. We report on the performance of this algorithm for a large number of case studies. In particular, the algorithm automatically verifies

[1] Under the name of *chemical reaction networks*.

many standard population protocols described in the literature [5,8,20,22,23,28,31], as well as liveness properties of distributed algorithms for leader election and mutual exclusion [3,40,42,44,50,59,61,64].

*Related Work.* The parameterized verification of replicated systems was first studied in [41], where they were modeled as counter systems. This allows one to apply many efficient techniques [11,24,37,47]. Most of these works are inherently designed for safety properties, and some can also handle fair termination [38], but none of them handles stochastic scheduling. To the best of our knowledge, the only works studying parameterized verification of liveness properties under our notion of stochastic scheduling are those on verification of population protocols. For *fixed* populations, protocols can be verified with standard probabilistic model checking [13,65], and early works follow this approach [28,31,60,63]. Subsequently, an algorithm and a tool for the *parameterized* verification of population protocols were described in [21,22], and a first version of stage graphs was introduced in [23] for analyzing the expected termination time of population protocols. In this paper we overhaul the framework of [23] for liveness verification, drawing inspiration from the safety verification technology of [21,22]. Compared to [21,22], our approach is not limited to a specific subclass of protocols, and captures models beyond population protocols. Furthermore, our new techniques for computing Presburger certificates subsume the procedure of [22]. In comparison to [23], we provide the first completeness and complexity results for stage graphs. Further, our stage graphs can prove correctness of population protocols and even more general liveness properties, while those of [23] can only prove termination. We also introduce novel techniques for computing stage graphs, which compared to [23] can greatly reduce their size and allows us to prove more examples correct.

There is also a large body of work on parameterized verification via cutoff techniques: one shows that a specification holds for any number of agents iff it holds for any number of agents below some threshold called the cutoff (see [6,26,30,34,46], and [16] for a comprehensive survey). Cut-off techniques can be applied to systems with an array or ring communication structure, but they require the existence and effectiveness of a cutoff, which is not the case in our setting. Further parameterized verification techniques are regular model checking [1,25] and automata learning [7]. The classes of communication structures they can handle are orthogonal to ours: arrays and rings for regular model checking and automata learning, and cliques in our work. Regular model checking and learning have recently been employed to verify safety properties [29], liveness properties under arbitrary schedulers [55] and termination under finitary fairness [51]. The classes of schedulers considered in [51,55] are incomparable to ours: arbitrary schedulers in [55], and finitary-fair schedulers in [51]. Further, these works are based on symbolic state-space exploration, while our techniques are based on automatic construction of invariants and ranking functions [16].

## 2 Preliminaries

Let $\mathbb{N}$ denote $\{0, 1, \ldots\}$ and let $E$ be a finite set. A *unordered vector* over $E$ is a mapping $V : E \to \mathbb{Z}$. In particular, a *multiset* over $E$ is an unordered vector $M : E \to \mathbb{N}$ where $M(e)$ denotes the number of occurrences of $e$ in $M$. The sets of all unordered vectors and multisets over $E$ are respectively denoted $\mathbb{Z}^E$ and $\mathbb{N}^E$. Vector addition, subtraction and comparison are defined componentwise. The *size* of a multiset $M$ is denoted $|M| = \sum_{e \in E} M(e)$. We let $E^{\langle k \rangle}$ denote the set of all multisets over $E$ of size $k$. We sometimes describe multisets using a set-like notation, e.g.$M = \wr f, g, g \wr$ or equivalently $M = \wr f, 2 \cdot g \wr$ is such that $M(f) = 1$, $M(g) = 2$ and $M(e) = 0$ for all $e \notin \{f, g\}$.

*Presburger Arithmetic.* Let $X$ be a set of variables. The set of formulas of *Presburger arithmetic* over $X$ is the result of closing atomic formulas, as defined in the next sentence, under Boolean operations and first-order existential quantification. Atomic formulas are of the form $\sum_{i=1}^{k} a_i x_i \sim b$, where $a_i$ and $b$ are integers, $x_i$ are variables and $\sim$ is either $<$ or $\equiv_m$, the latter denoting the congruence modulo $m$ for any $m \geq 2$. Formulas over $X$ are interpreted on $\mathbb{N}^X$. Given a formula $\phi$ of Presburger arithmetic, we let $[\![\phi]\!]$ denote the set of all multisets satisfying $\phi$. A set $E \subseteq \mathbb{N}^X$ is a *Presburger set* if $E = [\![\phi]\!]$ for some formula $\phi$.

### 2.1 Replicated Systems

A *replicated system* over $Q$ of arity $n$ is a tuple $\mathcal{P} = (Q, T)$, where $T \subseteq \bigcup_{k=0}^{n} Q^{\langle k \rangle} \times Q^{\langle k \rangle}$ is a *transition relation* containing the set of *silent* transitions $\bigcup_{k=0}^{n} \{(\boldsymbol{x}, \boldsymbol{x}) \mid \boldsymbol{x} \in Q^{\langle k \rangle})\}$[2]. A *configuration* is a multiset $C$ of states, which we interpret as a global state with $C(q)$ agents in each state $q \in Q$.

For every $t = (\boldsymbol{x}, \boldsymbol{y}) \in T$ with $\boldsymbol{x} = \wr X_1, X_2, \ldots, X_k \wr$ and $\boldsymbol{y} = \wr Y_1, Y_2, \ldots, Y_k \wr$, we write $X_1 X_2 \cdots X_k \mapsto Y_1 Y_2 \cdots Y_k$ and let ${}^\bullet t \overset{\text{def}}{=} \boldsymbol{x}$, $t^\bullet \overset{\text{def}}{=} \boldsymbol{y}$ and $\Delta(t) \overset{\text{def}}{=} t^\bullet - {}^\bullet t$. A transition $t$ is *enabled* at a configuration $C$ if $C \geq {}^\bullet t$ and, if so, can *occur*, leading to the configuration $C' = C + \Delta(t)$. If $t$ is not enabled at $C$, then we say that it is *disabled*. We use the following reachability notation:

$$\begin{aligned}
C \xrightarrow{t} C' &\iff t \text{ is enabled at } C \text{ and its occurrence leads to } C', \\
C \rightarrow C' &\iff C \xrightarrow{t} C' \text{ for some } t \in T, \\
C \xrightarrow{w} C' &\iff C = C_0 \xrightarrow{w_1} C_1 \cdots \xrightarrow{w_n} C_n = C' \text{ for some } C_0, C_1, \ldots, C_n \in \mathbb{N}^Q, \\
C \xrightarrow{*} C' &\iff C \xrightarrow{w} C' \text{ for some } w \in T^*.
\end{aligned}$$

Observe that, by definition of transitions, $C \rightarrow C'$ implies $|C| = |C'|$, and likewise for $C \xrightarrow{*} C'$. Intuitively, transitions cannot create or destroy agents.

A *run* is an infinite sequence $C_0 t_1 C_1 t_2 C_2 \cdots$ such that $C_i \xrightarrow{t_{i+1}} C_{i+1}$ for every $i \geq 0$. Given $L \subseteq T^*$ and a set of configurations $\mathcal{C}$, we let

$$\begin{aligned}
\mathit{post}_L(\mathcal{C}) &\overset{\text{def}}{=} \{C' : C \in \mathcal{C}, w \in L, C \xrightarrow{w} C'\}, & \mathit{post}^*(\mathcal{C}) &\overset{\text{def}}{=} \mathit{post}_{T^*}(\mathcal{C}), \\
\mathit{pre}_L(\mathcal{C}) &\overset{\text{def}}{=} \{C : C' \in \mathcal{C}, w \in L, C \xrightarrow{w} C'\}, & \mathit{pre}^*(\mathcal{C}) &\overset{\text{def}}{=} \mathit{pre}_{T^*}(\mathcal{C}).
\end{aligned}$$

[2] In the paper, we will omit the silent transitions when giving replicated systems.

*Stochastic Scheduling.* We assume that, given a configuration $C$, a probabilistic scheduler picks one of the transitions enabled at $C$. We only make the following two assumptions about the random experiment determining the transition: first, the probability of a transition depends only on $C$, and, second, every transition enabled at $C$ has a nonzero probability of occurring. Since $C \xrightarrow{*} C'$ implies $|C| = |C'|$, the number of configurations reachable from any configuration $C$ is finite. Thus, for every configuration $C$, the semantics of $\mathcal{P}$ from $C$ is a finite-state Markov chain rooted at $C$.

*Example 1.* Consider the replicated system $\mathcal{P} = (Q, T)$ of arity 2 with states $Q = \{\mathrm{A_Y}, \mathrm{A_N}, \mathrm{P_Y}, \mathrm{P_N}\}$ and transitions $T = \{t_1, t_2, t_3, t_4\}$, where

$$t_1 \colon \mathrm{A_Y}\,\mathrm{A_N} \mapsto \mathrm{P_Y}\,\mathrm{P_N}, \qquad t_2 \colon \mathrm{A_Y}\,\mathrm{P_N} \mapsto \mathrm{A_Y}\,\mathrm{P_Y},$$
$$t_3 \colon \mathrm{A_N}\,\mathrm{P_Y} \mapsto \mathrm{A_N}\,\mathrm{P_N}, \qquad t_4 \colon \mathrm{P_Y}\,\mathrm{P_N} \mapsto \mathrm{P_N}\,\mathrm{P_N}.$$

Intuitively, at every moment in time, agents are either *Active* or *Passive*, and have output *Yes* or *No*, which corresponds to the four states of $Q$. This system is designed to satisfy the following property: for every configuration $C$ in which all agents are initially active, i.e., $C$ satisfies $C(\mathrm{P_Y}) = C(\mathrm{P_N}) = 0$, if $C(\mathrm{A_Y}) > C(\mathrm{A_N})$, then eventually all agents stay forever in the "yes" states $\{\mathrm{A_Y}, \mathrm{P_Y}\}$, and otherwise all agents eventually stay forever in the "no" states $\{\mathrm{A_N}, \mathrm{P_N}\}$. ◁

## 2.2 Qualitative Model Checking

Let us fix a replicated system $\mathcal{P} = (Q, T)$. Formulas of *linear temporal logic (LTL)* on $\mathcal{P}$ are defined by the following grammar:

$$\varphi \ ::= \phi \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mathbf{X}\varphi \mid \varphi \,\mathbf{U}\, \varphi$$

where $\phi$ is a Presburger formula over $Q$. We look at $\phi$ as an atomic proposition over the set $\mathbb{N}^Q$ of configurations. Formulas of LTL are interpreted over runs of $\mathcal{P}$ in the standard way. We abbreviate $\Diamond\varphi \equiv \mathit{true}\,\mathbf{U}\,\varphi$ and $\Box\varphi \equiv \neg\Diamond\neg\varphi$.

Let us now introduce the probabilistic interpretation of LTL. A configuration $C$ of $\mathcal{P}$ satisfies an LTL formula $\varphi$ *with probability* $p$ if $\Pr[C, \varphi] = p$, where $\Pr[C, \varphi]$ denotes the probability of the set of runs of $\mathcal{P}$ starting at $C$ that satisfy $\varphi$ in the finite-state Markov chain rooted at $C$. The measurability of this set of runs for every $C$ and $\varphi$ follows from well-known results [65]. The *qualitative model checking problem* consists of, given an LTL formula $\varphi$ and a set of configurations $\mathcal{I}$, deciding whether $\Pr[C, \varphi] = 1$ for every $C \in \mathcal{I}$. We will often work with the complement problem, i.e., deciding whether $\Pr[C, \neg\varphi] > 0$ for some $C \in \mathcal{I}$.

In contrast to the action-based qualitative model checking problem of [35], our version of the problem is undecidable due to adding atomic propositions over configurations (see the full version of the paper [19] for a proof):

**Theorem 1.** *The qualitative model checking problem is not semi-decidable.*

It is known that qualitative model checking problems of finite-state probabilistic systems reduces to model checking of non-probabilistic systems under an adequate notion of fairness.

**Definition 1.** *A run of a replicated system $\mathcal{P}$ is* fair *if for every possible step $C \xrightarrow{t} C'$ of $\mathcal{P}$ the following holds: if the run contains infinitely many occurrences of $C$, then it also contains infinitely many occurrences of $C\,t\,C'$.*

So, intuitively, if a run can execute a step infinitely often, it eventually will. It is readily seen that a fair run of a finite-state transition system eventually gets "trapped" in one of its bottom strongly connected components, and visits each of its states infinitely often. Hence, fair runs of a finite-state Markov chain have probability one. The following proposition was proved in [35] for a model slightly less general than replicated systems; the proof can be generalized without effort:

**Proposition 1** **([35, Prop. 7]).** *Let $\mathcal{P}$ be a replicated system, let $C$ be a configuration of $\mathcal{P}$, and let $\varphi$ be an LTL formula. It is the case that $\Pr[C, \varphi] = 1$ iff every fair run of $\mathcal{P}$ starting at $C$ satisfies $\varphi$.*

We implicitly use this proposition from now on. In particular, we define:

**Definition 2.** *A configuration $C$* satisfies *$\varphi$ with probability 1, or just* satisfies *$\varphi$, if every fair run starting at $C$ satisfies $\varphi$, denoted by $C \models \varphi$. We let $[\![\varphi]\!]$ denote the set of configurations satisfying $\varphi$. A set $\mathcal{C}$ of configurations* satisfies *$\varphi$ if $\mathcal{C} \subseteq [\![\varphi]\!]$, i.e., if $C \models \varphi$ for every $C \in \mathcal{C}$.*

*Liveness Specifications for Replicated Systems.* We focus on a specific class of temporal properties for which the qualitative model checking problem is decidable and which is large enough to formalize many important specifications. Using well-known automata-theoretic technology, this class can also be used to verify all properties describable in action-based LTL, see e.g. [35].

A *stable termination property* is given by a pair $\Pi = (\varphi_{\text{pre}}, \Phi_{post})$, where $\Phi_{post} = \{\varphi^1_{\text{post}}, \ldots, \varphi^k_{\text{post}}\}$ and $\varphi_{\text{pre}}, \varphi^1_{\text{post}}, \ldots, \varphi^k_{\text{post}}$ are Presburger formulas over $Q$ describing sets of configurations. Whenever $k = 1$, we sometimes simply write $\Pi = (\varphi_{\text{pre}}, \varphi_{\text{post}})$. The pair $\Pi$ induces the LTL property

$$\varphi_\Pi \stackrel{\text{def}}{=} \Diamond \bigvee_{i=1}^{k} \Box \varphi^i_{\text{post}}.$$

Abusing language, we say that a replicated system $\mathcal{P}$ *satisfies $\Pi$* if $[\![\varphi_{\text{pre}}]\!] \subseteq [\![\varphi_\Pi]\!]$, that is, if every configuration $C$ satisfying $\varphi_{\text{pre}}$ satisfies $\varphi_\Pi$ with probability 1. The *stable termination problem* is the qualitative model checking problem for $\mathcal{I} = [\![\varphi_{\text{pre}}]\!]$ and $\varphi = \varphi_\Pi$ given by a stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{post})$.

*Example 2.* Let us reconsider the system from Example 1. We can formally specify that all agents will eventually agree on the majority output *Yes* or *No*. Let $\Pi^{\text{Y}} = (\varphi^{\text{Y}}_{\text{pre}}, \varphi^{\text{Y}}_{\text{post}})$ and $\Pi^{\text{N}} = (\varphi^{\text{N}}_{\text{pre}}, \varphi^{\text{N}}_{\text{post}})$ be defined by:

$$\varphi^{\text{Y}}_{\text{pre}} = (\mathrm{A_Y} > \mathrm{A_N} \land \mathrm{P_Y} + \mathrm{P_N} = 0), \qquad \varphi^{\text{Y}}_{\text{post}} = (\mathrm{A_N} + \mathrm{P_N} = 0),$$
$$\varphi^{\text{N}}_{\text{pre}} = (\mathrm{A_Y} \leq \mathrm{A_N} \land \mathrm{P_Y} + \mathrm{P_N} = 0), \qquad \varphi^{\text{N}}_{\text{post}} = (\mathrm{A_Y} + \mathrm{P_Y} = 0).$$

The system satisfies the property specified in Example 1 iff it satisfies $\Pi^{\text{Y}}$ and $\Pi^{\text{N}}$. As an alternative (weaker) property, we could specify that the system always stabilizes to either output by $\Pi = (\varphi^{\text{Y}}_{\text{pre}} \lor \varphi^{\text{N}}_{\text{pre}}, \{\varphi^{\text{Y}}_{\text{post}}, \varphi^{\text{N}}_{\text{post}}\})$. ◁

## 3 Stage Graphs

In the rest of the paper, we fix a replicated system $\mathcal{P} = (Q, T)$ and a stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{post})$, where $\Phi_{post} = \{\varphi^1_{\text{post}}, \ldots, \varphi^k_{\text{post}}\}$, and address the problem of checking whether $\mathcal{P}$ satisfies $\Pi$. We start with some basic definitions on sets of configurations.

**Definition 3 (inductive sets, leads to, certificates)**

- *A set of configurations $\mathcal{C}$ is* inductive *if $C \in \mathcal{C}$ and $C \to C'$ implies $C' \in \mathcal{C}$.*
- *Let $\mathcal{C}, \mathcal{C}'$ be sets of configurations. We say that $\mathcal{C}$* leads to *$\mathcal{C}'$, denoted $\mathcal{C} \leadsto \mathcal{C}'$, if for all $C \in \mathcal{C}$, every fair run from $C$ eventually visits a configuration of $\mathcal{C}'$.*
- *A* certificate *for $\mathcal{C} \leadsto \mathcal{C}'$ is a function $f : \mathcal{C} \to \mathbb{N}$ satisfying that for every $C \in \mathcal{C} \setminus \mathcal{C}'$, there exists an execution $C \xrightarrow{*} C'$ such that $f(C) > f(C')$.*

Note that certificates only require the existence of some executions decreasing $f$, not for all of them to to decrease it. Despite this, we have:

**Proposition 2.** *For all inductive sets $\mathcal{C}, \mathcal{C}'$ of configurations, it is the case that: $\mathcal{C}$ leads to $\mathcal{C}'$ iff there exists a certificate for $\mathcal{C} \leadsto \mathcal{C}'$.*

The proof, which can be found in the full version [19], depends on two properties of replicated systems with stochastic scheduling. First, every configuration has only finitely many descendants. Second, for every fair run and for every finite execution $C \xrightarrow{w} C'$, if $C$ appears infinitely often in the run, then the run contains infinitely many occurrences of $C \xrightarrow{w} C'$. We can now introduce stage graphs:

**Definition 4 (stage graph).** *A* stage graph *of $\mathcal{P}$ for the property $\Pi$ is a directed acyclic graph whose nodes, called* stages, *are sets of configurations satisfying the following conditions:*

1. *every stage is an inductive set;*
2. *every configuration of $[\![\varphi_{\text{pre}}]\!]$ belongs to some stage;*
3. *if $\mathcal{C}$ is a non-terminal stage with successors $\mathcal{C}_1, \ldots, \mathcal{C}_n$, then there exists a certificate for $\mathcal{C} \leadsto (\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_n)$;*
4. *if $\mathcal{C}$ is a terminal stage, then $\mathcal{C} \models \varphi^i_{\text{post}}$ for some $i$.*

The existence of a stage graph implies that $\mathcal{P}$ satisfies $\Pi$. Indeed, by conditions 2–3 and repeated application of Proposition 2, every run starting at a configuration of $[\![\varphi_{\text{pre}}]\!]$ eventually reaches a terminal stage, say $\mathcal{C}$, and, by condition 1, stays in $\mathcal{C}$ forever. Since, by condition 4, all configurations of $\mathcal{C}$ satisfy some $\varphi^i_{\text{post}}$, after its first visit to $\mathcal{C}$ every configuration satisfies $\varphi^i_{\text{post}}$.

*Example 3.* Figure 1 depicts stage graphs for the system of Example 1 and the properties defined in Example 2. The reader can easily show that every stage $\mathcal{C}$ is inductive by checking that for every $C \in \mathcal{C}$ and every transition $t \in \{t_1, \ldots, t_4\}$ enabled at $C$, the step $C \xrightarrow{t_i} C'$ satisfies $C' \in \mathcal{C}$. For example, if a configuration satisfies $\mathrm{A_Y} > \mathrm{A_N}$, so does any successor configuration. $\triangleleft$

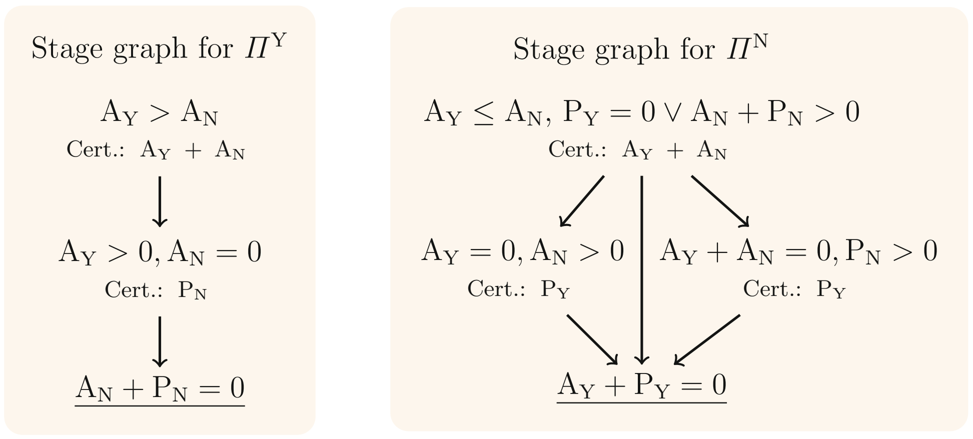

Stage graph for $\mathit{\Pi}^{\mathrm{Y}}$
$\mathrm{A_Y} > \mathrm{A_N}$
Cert.: $\mathrm{A_Y} + \mathrm{A_N}$
$\mathrm{A_Y} > 0, \mathrm{A_N} = 0$
Cert.: $\mathrm{P_N}$
$\underline{\mathrm{A_N} + \mathrm{P_N} = 0}$
Stage graph for $\mathit{\Pi}^{\mathrm{N}}$
$\mathrm{A_Y} \leq \mathrm{A_N}, \mathrm{P_Y} = 0 \vee \mathrm{A_N} + \mathrm{P_N} > 0$
Cert.: $\mathrm{A_Y} + \mathrm{A_N}$
$\mathrm{A_Y} = 0, \mathrm{A_N} > 0$
Cert.: $\mathrm{P_Y}$
$\mathrm{A_Y} + \mathrm{A_N} = 0, \mathrm{P_N} > 0$
Cert.: $\mathrm{P_Y}$
$\underline{\mathrm{A_Y} + \mathrm{P_Y} = 0}$

**Fig. 1.** Stage graphs for the system of Example 1.

The following proposition shows that stage graphs are a sound and complete technique for proving stable termination properties.

**Proposition 3.** *System $\mathcal{P}$ satisfies $\mathit{\Pi}$ iff it has a stage graph for $\mathit{\Pi}$.*

Proposition 3 does not tell us anything about the decidability of the stable termination problem. To prove that the problem is decidable, we introduce Presburger stage graphs. Intuitively these are stage graphs whose stages and certificates can be expressed by formulas of Presburger arithmetic.

**Definition 5 (Presburger stage graphs)**

– *A stage $\mathcal{C}$ is* Presburger *if $\mathcal{C} = [\![\phi]\!]$ for some Presburger formula $\phi$.*
– *A* bounded certificate *for $\mathcal{C} \rightsquigarrow \mathcal{C}'$ is a pair $(f, k)$, where $f \colon \mathcal{C} \to \mathbb{N}$ and $k \in \mathbb{N}$, satisfying that for every $C \in \mathcal{C} \setminus \mathcal{C}'$, there exists an execution $C \xrightarrow{w} C'$ such that $f(C) > f(C')$ and $|w| \leq k$.*
– *A* Presburger certificate *is a bounded certificate $(f, k)$ satisfying $f(C) = n \iff \varphi(C, n)$ for some Presburger formula $\varphi(\boldsymbol{x}, y)$.*
– *A* Presburger stage graph *is a stage graph whose stages and certificates are all Presburger.*

Using a powerful result from [36], we show that: (1) $\mathcal{P}$ satisfies $\mathit{\Pi}$ iff it has a Presburger stage graph for $\mathit{\Pi}$ (Theorem 2); (2) there exists a denumerable set of candidates for a Presburger stage graph for $\mathit{\Pi}$; and (3) there is an algorithm that decides whether a given candidate is a Presburger stage graph for $\mathit{\Pi}$ (Theorem 3). Together, (1–3) show that the stable termination problem is semi-decidable. To obtain decidability, we observe that the complement of the stable termination problem is also semi-decidable. Indeed, it suffices to enumerate all initial configurations $C \models \varphi_{\text{pre}}$, build for each such $C$ the (finite) graph $G_C$ of configurations reachable from $C$, and check if some bottom strongly connected component $\mathcal{B}$ of $G_C$ satisfies $\mathcal{B} \not\models \varphi^i_{\text{post}}$ for all $i$. This is the case iff some fair run starting at $C$ visits and stays in $\mathcal{B}$, which in turn is the case iff $\mathcal{P}$ violates $\mathit{\Pi}$.

**Theorem 2.** *System $\mathcal{P}$ satisfies $\Pi$ iff it has a Presburger stage graph for $\Pi$.*

We observe that testing whether a given graph is a Presburger stage graph reduces to Presburger arithmetic satisfiability, which is decidable [62] and whose complexity lies between 2-NEXP and 2-EXPSPACE [15]:

**Theorem 3.** *The problem of deciding whether an acyclic graph of Presburger sets and Presburger certificates is a Presburger stage graph, for a given stable termination property, is reducible in polynomial time to the satisfiability problem for Presburger arithmetic.*

## 4 Algorithmic Construction of Stage Graphs

At the current state of our knowledge, the decision procedure derived from Theorem 3 has little practical relevance. From a theoretical point of view, the TOWER-hardness result of [33] implies that the stage graph may have non-elementary size in the system size. In practice, systems have relatively small stage graphs, but, even so, the enumeration of all candidates immediately leads to a prohibitive combinatorial explosion.

For this reason, we present a procedure to automatically *construct* (not guess) a Presburger stage graph $G$ for a given replicated system $\mathcal{P}$ and a stable termination property $\Pi = (\varphi_{\text{pre}}, \Phi_{post})$. The procedure may *fail*, but, as shown in the experimental section, it succeeds for many systems from the literature.

The procedure is designed to be implemented on top of a solver for the existential fragment of Presburger arithmetic. While every formula of Presburger arithmetic has an equivalent formula within the existential fragment [32,62], quantifier-elimination may lead to a doubly-exponential blow-up in the size of the formula. Thus, it is important to emphasize that our procedure *never requires to eliminate quantifiers*: If the pre- and postconditions of $\Pi$ are supplied as quantifier-free formulas, then all constraints of the procedure remain in the existential fragment.

We give a high-level view of the procedure (see Algorithm 1), which uses several functions, described in detail in the rest of the paper. The procedure maintains a workset *WS* of Presburger stages, represented by existential Presburger formulas. Initially, the only stage is an inductive Presburger overapproximation $\mathit{PotReach}(\llbracket\varphi_{\text{pre}}\rrbracket)$ of the configurations reachable from $\llbracket\varphi_{\text{pre}}\rrbracket$ (*PotReach* is an abbreviation for "potentially reachable"). Notice that we must necessarily use an overapproximation, since $\mathit{post}^*(\llbracket\varphi_{\text{pre}}\rrbracket)$ is not always expressible in Presburger arithmetic[3]. We use a refinement of the overapproximation introduced in [22,37], equivalent to the overapproximation of [24].

In its main loop (lines 2–9), Algorithm 1 picks a Presburger stage $\mathcal{S}$ from the workset, and processes it. First, it calls $\text{Terminal}(\mathcal{S}, \Phi_{post})$ to check if $\mathcal{S}$ is terminal, i.e., whether $\mathcal{S} \models \varphi^i_{\text{post}}$ for some $\varphi^i_{\text{post}} \in \Phi_{post}$. This reduces to checking

[3] This follows easily from the fact that $\mathit{post}^*(\psi)$ is not always expressible in Presburger arithmetic for vector addition systems, even if $\psi$ denotes a single configuration [43].

**Algorithm 1:** procedure for the construction of stage graphs.

```
Input: replicated system $\mathcal{P} = (Q, T)$, stable term. property $\Pi = (\varphi_{\text{pre}}, \Phi_{post})$
Result: a stage graph of $\mathcal{P}$ for $\Pi$
1 $WS \leftarrow \{PotReach(\llbracket \varphi_{\text{pre}} \rrbracket)\}$
2 while $WS \neq \emptyset$ do
3     remove $\mathcal{S}$ from $WS$
4     if $\neg Terminal(\mathcal{S}, \Phi_{post})$ then
5         $U \leftarrow AsDead(\mathcal{S})$
6         if $U \neq \emptyset$ then
7             $WS \leftarrow WS \cup \{IndOverapprox(\mathcal{S}, U)\}$
8         else
9             $WS \leftarrow WS \cup Split(\mathcal{S})$
```

the unsatisfiability of the existential Presburger formula $\phi \wedge \neg\varphi^i_{\text{post}}$, where $\phi$ is the formula characterizing $\mathcal{S}$. If $\mathcal{S}$ is not terminal, then the procedure attempts to construct successor stages in lines 5–9, with the help of three further functions: *AsDead*, *IndOverapprox*, and *Split*. In the rest of this section, we present the intuition behind lines 5–9, and the specification of the three functions. Sections 5, 6 and 7 present the implementations we use for these functions.

Lines 5–9 are inspired by the behavior of most replicated systems designed by humans, and are based on the notion of *dead* transitions, which can never occur again (to be formally defined below). Replicated systems are usually designed to run in *phases*. Initially, all transitions are alive, and the end of a phase is marked by the "death" of one or more transitions, i.e., by reaching a configuration at which these transitions are dead. The system keeps "killing transitions" until no transition that is still alive can lead to a configuration violating the postcondition. The procedure mimics this pattern. It constructs stage graphs in which if $\mathcal{S}'$ is a successor of $\mathcal{S}$, then the set of transitions dead at $\mathcal{S}'$ is a *proper superset* of the transitions dead at $\mathcal{S}$. For this, $AsDead(\mathcal{S})$ computes a set of transitions that are alive at some configuration of $\mathcal{S}$, but which will become dead in every fair run starting at $\mathcal{S}$ (line 5). Formally, $AsDead(\mathcal{S})$ returns a set $U \subseteq \overline{Dead(\mathcal{S})}$ such that $\mathcal{S} \models \Diamond\text{dead}(U)$, defined as follows.

**Definition 6.** *A transition of a replicated system $\mathcal{P}$ is* dead *at a configuration $C$ if it is disabled at every configuration reachable from $C$ (including $C$ itself). A transition is* dead *at a stage $\mathcal{S}$ if it is dead at every configuration of $\mathcal{S}$. Given a stage $\mathcal{S}$ and a set $U$ of transitions, we use the following notations:*

- *$Dead(\mathcal{S})$: the set of transitions dead at $\mathcal{S}$;*
- *$\llbracket dis(U) \rrbracket$: the set of configurations at which all transitions of $U$ are disabled;*
- *$\llbracket dead(U) \rrbracket$: the set of configurations at which all transitions of $U$ are dead.*

Observe that we can compute $Dead(\mathcal{S})$ by checking unsatisfiability of a sequence of existential Presburger formulas: as $\mathcal{S}$ is inductive, we have $Dead(\mathcal{S}) =$

$\{t \mid \mathcal{S} \models \text{dis}(t)\}$, and $\mathcal{S} \models \text{dis}(t)$ holds iff the existential Presburger formula $\exists C \colon \phi(C) \wedge C \geq {}^\bullet t$ is unsatisfiable, where $\phi$ is the formula characterizing $\mathcal{S}$.

The following proposition, whose proof appears in the full version [19], shows that determining whether a given transition will eventually become dead, while decidable, is PSPACE-hard. Therefore, Sect. 7 describes two implementations of this function, and a way to combine them, which exhibit a good trade-off between precision and computation time.

**Proposition 4.** *Given a replicated system $\mathcal{P}$, a stage $\mathcal{S}$ represented by an existential Presburger formula $\phi$ and a set of transitions $U$, determining whether $\mathcal{S} \models \Diamond dead(U)$ holds is decidable and PSPACE-hard.*

If the set $U$ returned by $AsDead(\mathcal{S})$ is nonempty, then we know that every fair run starting at a configuration of $\mathcal{S}$ will eventually reach a configuration of $\mathcal{S} \cap [\![\text{dead}(U)]\!]$. So, this set, or any inductive overapproximation of it, can be a legal successor of $\mathcal{S}$ in the stage graph. Function $IndOverapprox(\mathcal{S}, U)$ returns such an inductive overapproximation (line 7). To be precise, we show in Sect. 5 that $[\![\text{dead}(U)]\!]$ is a Presburger set that can be computed exactly, albeit in doubly-exponential time in the worst case. The section also shows how to compute overapproximations more efficiently. If the set $U$ returned by $AsDead(\mathcal{S})$ is empty, then we cannot yet construct any successor of $\mathcal{S}$. Indeed, recall that we want to construct stage graphs in which if $\mathcal{S}'$ is a successor of $\mathcal{S}$, then $Dead(\mathcal{S}')$ is a *proper superset* of $Dead(\mathcal{S})$. In this case, we proceed differently and try to split $\mathcal{S}$:

**Definition 7.** *A* split *of some stage $\mathcal{S}$ is a set $\{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ of (not necessarily disjoint) stages such that the following holds:*

- *$Dead(\mathcal{S}_i) \supset Dead(\mathcal{S})$ for every $1 \leq i \leq k$, and*
- *$\mathcal{S} = \bigcup_{i=1}^{k} \mathcal{S}_i$.*

If there exists a split $\{\mathcal{S}_1, \ldots, \mathcal{S}_k\}$ of $\mathcal{S}$, then we can let $\mathcal{S}_1, \ldots, \mathcal{S}_k$ be the successors of $\mathcal{S}$ in the stage graph. Observe that a stage may indeed have a split. We have $Dead(\mathcal{C}_1 \cup \mathcal{C}_2) = Dead(\mathcal{C}_1) \cap Dead(\mathcal{C}_2)$, and hence $Dead(\mathcal{C}_1 \cup \mathcal{C}_2)$ may be a proper subset of both $Dead(\mathcal{C}_1)$ and $Dead(\mathcal{C}_2)$:

*Example 4.* Consider the system with states $\{q_1, q_2\}$ and transitions $t_i \colon q_i \mapsto q_i$ for $i \in \{1, 2\}$. Let $\mathcal{S} = \{C \mid C(q_1) = 0 \vee C(q_2) = 0\}$, i.e., $\mathcal{S}$ is the (inductive) stage of configurations disabling either $t_1$ or $t_2$. The set $\{\mathcal{S}_1, \mathcal{S}_2\}$, where $\mathcal{S}_i = \{C \in \mathcal{S} \mid C(q_i) = 0\}$, is a split of $\mathcal{S}$ satisfying $Dead(\mathcal{S}_i) = \{t_i\} \supset \emptyset = Dead(\mathcal{S})$. $\triangleleft$

The canonical split of $\mathcal{S}$, if it exists, is the set $\{\mathcal{S} \cap [\![\text{dead}(t)]\!] \mid t \notin Dead(\mathcal{S})\}$. As mentioned above, Sect. 5 shows that $[\![\text{dead}(U)]\!]$ can be computed exactly for every $U$, but the computation can be expensive. Hence, the canonical split can be computed exactly at potentially high cost. Our implementation uses an underapproximation of $[\![\text{dead}(t)]\!]$, described in Sect. 6.

## 5 Computing and Approximating $[\![\mathrm{dead}(U)]\!]$

We show that, given a set $U$ of transitions,

– we can effectively compute an existential Presburger formula describing the set $[\![\mathrm{dead}(U)]\!]$, with high computational cost in the worst case, and
– we can effectively compute constraints that overapproximate or underapproximate $[\![\mathrm{dead}(U)]\!]$, at a reduced computational cost.

**Downward and Upward Closed Sets.** We enrich $\mathbb{N}$ with the limit element $\omega$ in the usual way. In particular, $n < \omega$ holds for every $n \in \mathbb{N}$. An *$\omega$-configuration* is a mapping $C^\omega \colon Q \to \mathbb{N} \cup \{\omega\}$. The *upward closure* and *downward closure* of a set $\mathcal{C}^\omega$ of $\omega$-configurations are the sets of configurations $\uparrow\mathcal{C}^\omega$ and $\downarrow\mathcal{C}^\omega$, respectively defined as:

$$\uparrow\mathcal{C}^\omega \overset{\text{def}}{=} \{C \in \mathbb{N}^Q \mid C \geq C^\omega \text{ for some } C^\omega \in \mathcal{C}^\omega\},$$
$$\downarrow\mathcal{C}^\omega \overset{\text{def}}{=} \{C \in \mathbb{N}^Q \mid C \leq C^\omega \text{ for some } C^\omega \in \mathcal{C}^\omega\}.$$

A set $\mathcal{C}$ of configurations is *upward closed* if $\mathcal{C} = \uparrow\mathcal{C}$, and *downward closed* if $\mathcal{C} = \downarrow\mathcal{C}$. These facts are well-known from the theory of well-quasi orderings:

**Lemma 1.** *For every set $\mathcal{C}$ of configurations, the following holds:*

1. *$\mathcal{C}$ is upward closed iff $\overline{\mathcal{C}}$ is downward closed (and vice versa);*
2. *if $\mathcal{C}$ is upward closed, then there is a unique minimal finite set of configurations $\mathit{inf}(\mathcal{C})$, called its* basis, *such that $\mathcal{C} = \uparrow \mathit{inf}(\mathcal{C})$;*
3. *if $\mathcal{C}$ is downward closed, then there is a unique minimal finite set of $\omega$-configurations $\mathit{sup}(\mathcal{C})$, called its* decomposition, *such that $\mathcal{C} = \downarrow \mathit{sup}(\mathcal{C})$.*

**Computing $[\![\mathrm{dead}(U)]\!]$ Exactly.** It follows immediately from Definition 6 that both $[\![\mathrm{dis}(U)]\!]$ and $[\![\mathrm{dead}(U)]\!]$ are downward closed. Indeed, if all transitions of $U$ are disabled at $C$, and $C' \leq C$, then they are also disabled at $C'$, and clearly the same holds for transitions dead at $C$. Furthermore:

**Proposition 5.** *For every set $U$ of transitions, the (downward) decomposition of both $\mathit{sup}([\![\mathit{dis}(U)]\!])$ and $\mathit{sup}([\![\mathit{dead}(U)]\!])$ is effectively computable.*

*Proof.* For every $t \in U$ and $q \in {}^\bullet t$, let $C^\omega_{t,q}$ be the $\omega$-configuration such that $C^\omega_{t,q}(q) = {}^\bullet t(q) - 1$ and $C^\omega_{t,q}(p) = \omega$ for every $p \in Q \setminus \{q\}$. In other words, $C^\omega_{t,q}$ is the $\omega$-configuration made only of $\omega$'s except for state $q$ which falls short from ${}^\bullet t(q)$ by one. This $\omega$-configurations captures all configurations disabled in $t$ due to an insufficient amount of agents in state $q$. We have:

$$\sup([\![\mathrm{dis}(U)]\!]) = \{C^\omega_{t,q} : t \in U, q \in {}^\bullet t\}.$$

The latter can be made minimal by removing superfluous $\omega$-configurations.

For the case of $\sup([\![\mathrm{dead}(U)]\!])$, we invoke [45, Prop. 2] which gives a proof for the more general setting of (possibly unbounded) Petri nets. Their procedure is based on the well-known backwards reachability algorithm (see, e.g., [2,39]). □

Since $\sup([\![\text{dead}(U)]\!])$ is finite, its computation allows to describe $[\![\text{dead}(U)]\!]$ by the following linear constraint[4]:

$$\bigvee_{C^\omega \in \sup([\![\text{dead}(U)]\!])} \bigwedge_{q \in Q} [C(q) \leq C^\omega(q)] .$$

However, the cardinality of $\sup([\![\text{dead}(U)]\!])$ can be exponential [45, Remark for Prop. 2] in the system size. For this reason, we are interested in constructing both under- and over-approximations.

**Overapproximations of $[\![\text{dead}(\boldsymbol{U})]\!]$.** For every $i \in \mathbb{N}$, define $[\![\text{dead}(U)]\!]^i$ as:

$$[\![\text{dead}(U)]\!]^0 \overset{\text{def}}{=} [\![\text{dis}(U)]\!] \quad \text{and} \quad [\![\text{dead}(U)]\!]^{i+1} \overset{\text{def}}{=} \overline{\mathit{pre}_T(\overline{[\![\text{dead}(U)]\!]^i})} \cap [\![\text{dis}(U)]\!].$$

Loosely speaking, $[\![\text{dead}(U)]\!]^i$ is the set of configurations $C$ such that every configuration reachable in at most $i$ steps from $C$ disables $U$. We immediately have:

$$[\![\text{dead}(U)]\!] = \bigcap_{i=0}^{\infty} [\![\text{dead}(U)]\!]^i.$$

Using Proposition 5 and the following proposition, we obtain that $[\![\text{dead}(U)]\!]^i$ is an effectively computable overapproximation of $[\![\text{dead}(U)]\!]$.

**Proposition 6.** *For every Presburger set $\mathcal{C}$ and every set of transitions $U$, the sets $\mathit{pre}_U(\mathcal{C})$ and $\mathit{post}_U(\mathcal{C})$ are effectively Presburger.*

Recall that function $\mathit{IndOverapprox}(\mathcal{S}, U)$ of Algorithm 1 must return an *inductive* overapproximation of $[\![\text{dead}(U)]\!]$. Since $[\![\text{dead}(U)]\!]^i$ might not be inductive in general, our implementation uses either the inductive overapproximations $\mathit{IndOverapprox}^i(\mathcal{S}, U) \overset{\text{def}}{=} \mathit{PotReach}(\mathcal{S} \cap [\![\text{dead}(U)]\!]^i)$, or the exact value $\mathit{IndOverapprox}^\infty(\mathcal{S}, U) \overset{\text{def}}{=} \mathcal{S} \cap [\![\text{dead}(U)]\!]$. The table of results in the experimental section describes for each benchmark which overapproximation was used.

**Underapproximations of $[\![\text{dead}(\boldsymbol{U})]\!]$: Death Certificates.** A *death certificate* for $U$ in $\mathcal{P}$ is a finite set $\mathcal{C}^\omega$ of $\omega$-configurations such that:

1. $\downarrow\mathcal{C}^\omega \models \text{dis}(U)$, i.e., every configuration of $\downarrow\mathcal{C}^\omega$ disables $U$, and
2. $\downarrow\mathcal{C}^\omega$ is inductive, i.e., $\mathit{post}_T(\downarrow\mathcal{C}^\omega) \subseteq \downarrow\mathcal{C}^\omega$.

If $U$ is dead at a set $\mathcal{C}$ of configurations, then there is always a certificate that proves it, namely $\sup([\![\text{dead}(U)]\!])$. In particular, if $\mathcal{C}^\omega$ is a death certificate for $U$ then $\downarrow\mathcal{C}^\omega \subseteq [\![\text{dead}(U)]\!]$, that is, $\downarrow\mathcal{C}^\omega$ is an underapproximation of $[\![\text{dead}(U)]\!]$

Using Proposition 6, it is straightforward to express in Presburger arithmetic that a finite set $\mathcal{C}^\omega$ of $\omega$-configurations is a death certificate for $U$:

**Proposition 7.** *For every $k \geq 1$ there is an existential Presburger formula $\mathit{DeathCert}_k(U, \mathcal{C}^\omega)$ that holds iff $\mathcal{C}^\omega$ is a death certificate of size $k$ for $U$.*

[4] Observe that if $C^\omega(q) = \omega$, then the term "$C(q) \leq \omega$" is equivalent to "**true**".

## 6 Splitting a Stage

Given a stage $\mathcal{S}$, we try to find a set $\mathcal{C}_1^\omega, \dots, \mathcal{C}_\ell^\omega$ of death certificates for transitions $t_1, \dots, t_\ell \in T \setminus \mathit{Dead}(\mathcal{S})$ such that $\mathcal{S} \subseteq {\downarrow}\mathcal{C}_1^\omega \cup \dots \cup {\downarrow}\mathcal{C}_\ell^\omega$. This allows us to split $\mathcal{S}$ into $\mathcal{S}_1, \dots, \mathcal{S}_\ell$, where $\mathcal{S}_i \overset{\text{def}}{=} \mathcal{S} \cap {\downarrow}\mathcal{C}_i^\omega$.

For any fixed size $k \geq 1$ and any fixed $\ell$, we can find death certificates $\mathcal{C}_1^\omega, \dots, \mathcal{C}_\ell^\omega$ of size at most $k$ by solving a Presburger formula. However, the formula does not belong to the existential fragment, because the inclusion check $\mathcal{S} \subseteq {\downarrow}\mathcal{C}_1^\omega \cup \dots \cup {\downarrow}\mathcal{C}_\ell^\omega$ requires universal quantification. For this reason, we proceed iteratively. For every $i \geq 0$, after having found $\mathcal{C}_1^\omega, \dots, \mathcal{C}_i^\omega$ we search for a pair $(C_{i+1}, \mathcal{C}_{i+1}^\omega)$ such that

(i) $\mathcal{C}_{i+1}^\omega$ is a death certificate for some $t_{i+1} \in T \setminus \mathit{Dead}(\mathcal{S})$;
(ii) $C_{i+1} \in \mathcal{S} \cap {\downarrow}\mathcal{C}_{i+1}^\omega \setminus ({\downarrow}\mathcal{C}_1^\omega \cup \dots \cup {\downarrow}\mathcal{C}_i^\omega)$.

An efficient implementation requires to guide the search for $(C_{i+1}, \mathcal{C}_{i+1}^\omega)$, because otherwise the search procedure might not even terminate, or might split $\mathcal{S}$ into too many parts, blowing up the size of the stage graph. Our search procedure employs the following heuristic, which works well in practice. We only consider the case $k = 1$, and search for a pair $(C_{i+1}, C_{i+1}^\omega)$ satisfying (i) and (ii) above, and additionally:

(iii) all components of $C_{i+1}^\omega$ are either $\omega$ or between 0 and $\max_{t \in T, q \in Q} {}^\bullet t(q) - 1$;
(iv) for every $\omega$-configuration $C^\omega$, if $(C_{i+1}, C^\omega)$ satisfies (i)–(iii), then $C_{i+1}^\omega \leq C^\omega$;
(v) for every pair $(C, C^\omega)$, if $(C, C^\omega)$ satisfies (i)–(iv), then $C^\omega \leq C_{i+1}^\omega$.

Condition (iii) guarantees termination. Intuitively, condition (iv) leads to certificates valid for sets $U \subseteq T \setminus \mathit{Dead}(\mathcal{S})$ as large as possible. So it allows us to avoid splits that, loosely speaking, do not make as much progress as they could. Condition (v) allows us to avoid splits with many elements because each element of the split has a small intersection with $\mathcal{S}$.

An example illustrating these conditions is given in the full version [19].

## 7 Computing Eventually Dead Transitions

Recall that the function $\mathit{AsDead}(\mathcal{S})$ takes an inductive Presburger set $\mathcal{S}$ as input, and returns a (possibly empty) set $U \subseteq \overline{\mathit{Dead}(\mathcal{S})}$ of transitions such that $\mathcal{S} \models \Diamond\text{dead}(U)$. This guarantees $\mathcal{S} \leadsto [\![\text{dead}(U)]\!]$ and, since $\mathcal{S}$ is inductive, also $\mathcal{S} \leadsto \mathcal{S} \cap [\![\text{dead}(U)]\!]$.

By Proposition 4, deciding if there exists a non-empty set $U$ of transitions such that $\mathcal{S} \models \Diamond\text{dead}(U)$ holds is $\mathsf{PSPACE}$-hard, which makes a polynomial reduction to satisfiability of existential Presburger formulas unlikely. So we design incomplete implementations of $\mathit{AsDead}(\mathcal{S})$ with lower complexity. Combining these implementations, the lack of completeness essentially vanishes in practice.

The implementations are inspired by Proposition 2, which shows that $\mathcal{S} \leadsto [\![\text{dead}(U)]\!]$ holds iff there exists a certificate $f$ such that:

$$\forall C \in \mathcal{S} \setminus [\![\text{dead}(U)]\!] : \exists C \xrightarrow{*} C' : f(C) > f(C'). \qquad \text{(Cert)}$$

To find such certificates efficiently, we only search for *linear* functions $f(C) = \sum_{q \in Q} \boldsymbol{a}(q) \cdot C(q)$ with coefficients $\boldsymbol{a}(q) \in \mathbb{N}$ for each $q \in Q$.

## 7.1 First Implementation: Linear Ranking Functions

Our first procedure computes the existence of a linear *ranking function.*

**Definition 8.** *A function $r \colon \mathcal{S} \to \mathbb{N}$ is a ranking function for $\mathcal{S}$ and $U$ if for every $C \in \mathcal{S}$ and every step $C \xrightarrow{t} C'$ the following holds:*

*1. if $t \in U$, then $r(C) > r(C')$; and*
*2. if $t \notin U$, then $r(C) \geq r(C')$.*

**Proposition 8.** *If $r \colon \mathcal{S} \to \mathbb{N}$ is a ranking function for $\mathcal{S}$ and $U$, then there exists $k \in \mathbb{N}$ such that $(r, k)$ is a bounded certificate for $\mathcal{S} \rightsquigarrow [\![\mathit{dead}(U)]\!]$.*

*Proof.* Let $M$ be the minimal finite basis of the upward closed set $\overline{[\![\text{dead}(U)]\!]}$. For every configuration $D \in M$, let $\sigma_D$ be a shortest sequence that enables some transition of $t_D \in U$ from $D$, i.e., such that $D \xrightarrow{\sigma_D} D' \xrightarrow{t_D} D''$ for some $D'$, $D''$. Let $k \overset{\text{def}}{=} \max\{|\sigma_D t_D| : D \in M\}$.

Let $C \in \mathcal{S} \setminus [\![\text{dead}(U)]\!]$. Since $C \in \overline{[\![\text{dead}(U)]\!]}$, we have $C \geq D$ for some $D \in M$. By monotonicity, we have $C \xrightarrow{\sigma_D} C' \xrightarrow{t_D} C''$ for some configurations $C'$ and $C''$. By Definition 8, we have $r(C) \geq r(C') > r(C'')$, and so condition (Cert) holds. As $|\sigma_D t_D| \leq k$, we have that $(r, k)$ is a bounded certificate. $\square$

It follows immediately from Definition 8 that if $r_1$ and $r_2$ are ranking functions for sets $U_1$ and $U_2$ respectively, then $r$ defined as $r(C) \overset{\text{def}}{=} r_1(C) + r_2(C)$ is a ranking function for $U_1 \cup U_2$. Therefore, there exists a unique maximal set of transitions $U$ such that $\mathcal{S} \rightsquigarrow [\![\text{dead}(U)]\!]$ can be proved by means of a ranking function. Further, $U$ can be computed by collecting all transitions $t \in \overline{\mathit{Dead}(\mathcal{S})}$ such that there exists a ranking function $r_t$ for $\{t\}$. The existence of a *linear* ranking function $r_t$ can be decided in polynomial time via linear programming, as follows. Recall that for every step $C \xrightarrow{u} C'$, we have $C' = C + \Delta(u)$. So, by linearity, we have $r_t(C) \geq r_t(C') \iff r_t(C' - C) \leq 0 \iff r_t(\Delta(u)) \leq 0$. Thus, the constraints of Definition 8 can be specified as:

$$\boldsymbol{a} \cdot \Delta(t) < 0 \quad \wedge \bigwedge_{u \in \overline{\mathit{Dead}(\mathcal{S})}} \boldsymbol{a} \cdot \Delta(u) \leq 0,$$

where $\boldsymbol{a} \colon Q \to \mathbb{Q}_{\geq 0}$ gives the coefficients of $r_t$, that is, $r_t(C) = \boldsymbol{a} \cdot C$, and $\boldsymbol{a} \cdot \boldsymbol{x} \overset{\text{def}}{=} \sum_{q \in Q} \boldsymbol{a}(q) \cdot \boldsymbol{x}(q)$ for $\boldsymbol{x} \in \mathbb{N}^Q$. Observe that a solution may yield a function whose codomain differs from $\mathbb{N}$. However, this is not an issue since we can scale it with the least common denominator of each $\boldsymbol{a}(q)$.

### 7.2 Second Implementation: Layers

*Transitions layers* were introduced in [22] as a technique to find transitions that will eventually become dead. Intuitively, a set $U$ of transitions is a layer if (1) no run can contain only transitions of $U$, and (2) $U$ becomes dead once disabled; the first condition guarantees that $U$ eventually becomes disabled, and the second that it eventually becomes dead. We formalize layers in terms of *layer functions*.

**Definition 9.** *A function $\ell \colon \mathcal{S} \to \mathbb{N}$ is a* layer function *for $\mathcal{S}$ and $U$ if:*

**C1.** *$\ell(C) > \ell(C')$ for every $C \in \mathcal{S}$ and every step $C \xrightarrow{t} C'$ with $t \in U$; and*
**C2.** *$[\![dis(U)]\!] = [\![dead(U)]\!]$.*

**Proposition 9.** *If $\ell \colon \mathcal{S} \to \mathbb{N}$ is a layer function for $\mathcal{S}$ and $U$, then $(\ell, 1)$ is a bounded certificate for $\mathcal{S} \rightsquigarrow [\![dead(U)]\!]$.*

*Proof.* Let $C \in \mathcal{S} \setminus [\![\mathrm{dead}(U)]\!]$. By condition **C2**, we have $C \notin [\![\mathrm{dis}(U)]\!]$. So there exists a step $C \xrightarrow{u} C'$ where $u \in U$. By condition **C1**, we have $\ell(C) > \ell(C')$, so condition (Cert) holds and $(\ell, 1)$ is a bounded certificate.

Let $\mathcal{S}$ be a stage. For every set of transitions $U \subseteq \overline{Dead(\mathcal{S})}$ we can construct a Presburger formula *lin-layer*$(U, \boldsymbol{a})$ that holds iff there there exists a *linear* layer function for $U$, i.e., a layer function of the form $\ell(C) = \boldsymbol{a} \cdot C$ for a vector of coefficients $\boldsymbol{a} \colon Q \to \mathbb{Q}_{\geq 0}$. Condition **C1**, for a linear function $\ell(C)$, is expressed by the existential Presburger formula

$$\mathit{lin\text{-}layer\text{-}fun}(U, \boldsymbol{a}) \stackrel{\text{def}}{=} \bigwedge_{u \in U} \boldsymbol{a} \cdot \Delta(u) < 0.$$

Condition **C2** is expressible in Presburger arithmetic because of Proposition 5. However, instead of computing $[\![\mathrm{dead}(U)]\!]$ explicitly, there is a more efficient way to express this constraint. Intuitively, $[\![\mathrm{dis}(U)]\!] = [\![\mathrm{dead}(U)]\!]$ is the case if enabling a transition $u \in U$ requires to have previously enabled some transition $u' \in U$. This observation leads to:

**Proposition 10.** *A set $U$ of transitions satisfies $[\![dis(U)]\!] = [\![dead(U)]\!]$ iff it satisfies the existential Presburger formula*

$$\mathit{dis\text{-}eq\text{-}dead}(U) \stackrel{\mathit{def}}{=} \bigwedge_{t \in T} \bigwedge_{u \in U} \bigvee_{u' \in U} {}^{\bullet}t + ({}^{\bullet}u \ominus t^{\bullet}) \geq {}^{\bullet}u'$$

*where $\boldsymbol{x} \ominus \boldsymbol{y} \in \mathbb{N}^Q$ is defined by $(\boldsymbol{x} \ominus \boldsymbol{y})(q) \stackrel{\mathit{def}}{=} \max(\boldsymbol{x}(q) - \boldsymbol{y}(q), 0)$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^Q$.*

This allows us to give the constraint *lin-layer*$(U, \boldsymbol{a})$, which is of polynomial size:

$$\mathit{lin\text{-}layer}(U, \boldsymbol{a}) \stackrel{\text{def}}{=} \mathit{lin\text{-}layer\text{-}fun}(U, \boldsymbol{a}) \wedge \mathit{dis\text{-}eq\text{-}dead}(U).$$

### 7.3 Comparing Ranking and Layer Functions

The ranking and layer functions of Sects. 7.1 and 7.2 are incomparable in power, that is, there are sets of transitions for which a ranking function but no layer function exists, and vice versa. This is shown by the following two systems:

$$\begin{aligned}\mathcal{P}_1 &= (\{\mathrm{A},\mathrm{B},\mathrm{C}\}, \{t_1\colon \mathrm{A\,B} \mapsto \mathrm{C\,C},\ t_2\colon \mathrm{A} \mapsto \mathrm{B},\ t_3\colon \mathrm{B} \mapsto \mathrm{A}\}),\\ \mathcal{P}_2 &= (\{\mathrm{A},\mathrm{B}\}, \quad \{t_4\colon \mathrm{A\,B} \mapsto \mathrm{A\,A},\ t_5\colon \mathrm{A} \mapsto \mathrm{B}\}).\end{aligned}$$

Consider the system $\mathcal{P}_1$, and let $\mathcal{S} = \mathbb{N}^Q$, i.e., $\mathcal{S}$ contains all configurations. Transitions $t_2$ and $t_3$ never become dead at $\wr \mathrm{A} \wr$ and can thus never be included in any $U$. Transition $t_1$ eventually becomes dead, as shown by the linear ranking function $r(C) = C(\mathrm{A}) + C(\mathrm{B})$ for $U = \{t_1\}$. But for this $U$, the condition **C2** for layer functions is not satisfied, as $[\![\mathrm{dis}(U)]\!] \ni \wr \mathrm{A},\mathrm{A} \wr \xrightarrow{t_2} \wr \mathrm{A},\mathrm{B} \wr \notin [\![\mathrm{dis}(U)]\!]$, so $[\![\mathrm{dis}(U)]\!] \neq [\![\mathrm{dead}(U)]\!]$. Therefore no layer function exists for this $U$.

Consider now the system $\mathcal{P}_2$, again with $\mathcal{S} = \mathbb{N}^Q$, and let $U = \{t_5\}$. Once $t_5$ is disabled, there is no agent in A, so both $t_4$ and $t_5$ are dead. So $[\![\mathrm{dis}(U)]\!] = [\![\mathrm{dead}(U)]\!]$. The linear layer function $\ell(C) = C(\mathrm{A})$ satisfies *lin-layer-fun*$(U, \boldsymbol{a})$, showing that $U$ eventually becomes dead. As $C \xrightarrow{t_4 t_5} C$ for $C = \wr \mathrm{A},\mathrm{B} \wr$, there is no ranking function $r$ for this $U$, which would need to satisfy $r(C) < r(C)$.

For our implementation of *AsDead*$(\mathcal{S})$, we therefore combine both approaches. We first compute (in polynomial time) the unique maximal set $U$ for which there is a linear ranking function. If this $U$ is non-empty, we return it, and otherwise compute a set $U$ of maximal size for which there is a linear layer function.

## 8 Experimental Results

We implemented the procedure of Sect. 4 on top of the SMT solver *Z3* [57], and use the Owl [48] and HOA [12] libraries for translating LTL formulas. The resulting tool automatically constructs stage graphs that verify stable termination properties for replicated systems. We evaluated it on two sets of benchmarks, described below. The first set contains population protocols, and the second leader election and mutual exclusion algorithms. All tests where performed on a machine with an Intel Xeon CPU E5-2630 v4 @ 2.20 GHz and 8GB of RAM. The results are depicted in Fig. 2 and can be reproduced by the certified artifact [18]. For parametric families of replicated systems, we always report the largest instance that we were able to verify with a timeout of one hour. For *IndOverapprox*, from the approaches in Sect. 5, we use *IndOverapprox*$^0$ in the examples marked with * and *IndOverapprox*$^\infty$ otherwise. Almost all constructed stage graphs are a chain with at most 3 stages. The only exceptions are the stage graphs for the approximate majority protocols that contained a binary split and 5 stages. The size of the Presburger formulas increases with increasing size of the replicated system. In the worst case, this growth can be exponential. However, the growth is linear in all examples marked with *.

| **Population protocols** (correctness) | | | |
|---|---|---|---|
| **Parameters** | $\|Q\|$ | $\|T\|$ | **Time** |
| Broadcast [31,22] * | | | |
| | 2 | 1 | $< 1s$ |
| Majority (Example 1)[22] * | | | |
| | 4 | 4 | $< 1s$ |
| Majority [23, Ex. 3] * | | | |
| | 5 | 6 | $< 1s$ |
| Majority [5] ("fast & exact") | | | |
| $m$=13, $d$=1 | 16 | 136 | $4s$ |
| $m$=21, $d$=1 (TO: 23,1) | 24 | 300 | $466s$ |
| $m$=21, $d$=20 (TO: 23,22) | 62 | 1953 | $3301s$ |
| Flock-of-birds [28,22] *: $x \geq c$ | | | |
| $c = 20$ | 21 | 210 | $5s$ |
| $c = 40$ | 41 | 820 | $45s$ |
| $c = 60$ | 61 | 1830 | $341s$ |
| $c = 80$ (TO: $c = 90$) | 81 | 3240 | $1217s$ |
| Flock-of-birds [20, Sect. 3]: $x \geq c$ | | | |
| $c = 60$ | 8 | 18 | $15s$ |
| $c = 90$ | 9 | 21 | $271s$ |
| $c = 120$ (TO: $c = 127$) | 9 | 21 | $2551s$ |
| Flock-of-birds [31,22, *threshold-n*] *: $x \geq c$ | | | |
| $c = 10$ | 11 | 19 | $< 1s$ |
| $c = 15$ | 16 | 29 | $1s$ |
| $c = 20$ (TO: $c = 25$) | 21 | 39 | $18s$ |
| Threshold [8][22, $v_{\max}$=$c+1$] *: $\boldsymbol{a} \cdot \boldsymbol{x} \geq c$ | | | |
| $c = 2$ | 28 | 288 | $7s$ |
| $c = 4$ | 44 | 716 | $26s$ |
| $c = 6$ | 60 | 1336 | $107s$ |
| $c = 8$ (TO: $c = 10$) | 76 | 2148 | $1089s$ |
| Threshold [20] ("succinct"): $\boldsymbol{a} \cdot \boldsymbol{x} \geq c$ | | | |
| $c = 7$ | 13 | 37 | $2s$ |
| $c = 31$ | 17 | 55 | $11s$ |
| $c = 127$ | 21 | 73 | $158s$ |
| $c = 511$ (TO: $c = 1023$) | 25 | 91 | $2659s$ |
| Remainder [22] *: $\boldsymbol{a} \cdot \boldsymbol{x} \equiv_m c$ | | | |
| $m = 5$ | 7 | 20 | $< 1s$ |
| $m = 15$ | 17 | 135 | $34s$ |
| $m = 20$ (TO: $m = 25$) | 22 | 230 | $1646s$ |

| **Population protocols** (stable cons.) | | | |
|---|---|---|---|
| **Parameters** | $\|Q\|$ | $\|T\|$ | **Time** |
| Approx. majority [27] (Cell cycle sw.) * | | | |
| | 3 | 4 | $< 1s$ |
| Approx. majority [51] (Coin game) * | | | |
| $k = 3$ | 2 | 4 | $< 1s$ |
| Approx. majority [56] (Moran proc.) * | | | |
| | 2 | 2 | $< 1s$ |

| **Leader election/Mutex algorithms** | | | |
|---|---|---|---|
| **Processes** | $\|Q\|$ | $\|T\|$ | **Time** |
| Leader election [44] (Israeli-Jalfon) | | | |
| 20 | 40 | 80 | $7s$ |
| 60 | 120 | 240 | $1493s$ |
| 70 (TO: 80) | 140 | 280 | $3295s$ |
| Leader election [42] (Herman) | | | |
| 21 | 42 | 42 | $9s$ |
| 51 | 102 | 102 | $300s$ |
| 81 (TO: 91) | 162 | 162 | $2800s$ |
| Mutex [40] (Array) | | | |
| 2 | 15 | 95 | $2s$ |
| 5 | 33 | 239 | $5s$ |
| 10 (TO: 11) | 63 | 479 | $938s$ |
| Mutex [59] (Burns) | | | |
| 2 | 11 | 75 | $1s$ |
| 4 | 19 | 199 | $119s$ |
| 5 (TO: 6) | 23 | 279 | $2232s$ |
| Mutex [3] (Dijkstra) | | | |
| 2 | 19 | 196 | $66s$ |
| 3 (TO: 4) | 27 | 488 | $3468s$ |
| Mutex [50] (Lehmann Rabin) | | | |
| 2 | 19 | 135 | $3s$ |
| 5 | 43 | 339 | $115s$ |
| 9 (TO: 10) | 75 | 611 | $2470s$ |
| Mutex [61] (Peterson) | | | |
| 2 | 13 | 86 | $2s$ |
| Mutex [64] (Szymanski) | | | |
| 2 | 17 | 211 | $10s$ |
| 3 (TO: 4) | 24 | 895 | $667s$ |

**Fig. 2.** Columns $|Q|$, $|T|$, and **Time** give the number of states and non-silent transitions, and the time for verification. Population protocols are verified for an infinite set of configurations. For parametric families, the smallest instance that could not be verified within one hour is shown in brackets, e.g. (TO: $c = 90$). Leader election and mutex algorithms are verified for one configuration. The number of processes leading to a timeout is given in brackets, e.g. (TO: 10).

*Population Protocols.* Population protocols [8,9] are replicated systems that compute Presburger predicates following the computation-as-consensus paradigm [10]. Depending on whether the initial configuration of agents satisfies the predicate or not, the agents of a correct protocol eventually agree on the output "yes" or "no", almost surely. Example 1 can be interpreted as a population protocol for the majority predicate $\mathrm{A_Y} > \mathrm{A_N}$, and the two stable termination properties that verify its correctness are described in Example 2. To show that a population protocol correctly computes a given predicate, we thus construct two Presburger stage graphs for the two corresponding stable termination properties. In all these examples, correctness is proved for an infinite set of initial configurations.

Our set of benchmarks contains a broadcast protocol [31], three majority protocols (Example 1, [23, Ex. 3], [5]), and multiple instances of parameterized families of protocols, where each protocol computes a different instance of a parameterized family of predicates[5]. These include various *flock-of-birds* protocol families ([28], [20, Sect. 3], [31, *threshold-n*]) for the family of predicates $x \geq c$ for some constant $c \geq 0$; two families for threshold predicates of the form $\boldsymbol{a} \cdot \boldsymbol{x} \geq c$ [8,20]; and one family for remainder protocols of the form $\boldsymbol{a} \cdot \boldsymbol{x} \equiv_m c$ [22]. Further, we check approximate majority protocols ([27,56], [51, *coin game*]). As these protocols only compute the predicate with large probability but not almost surely, we only verify that they always converge to a stable consensus.

*Comparison with* [22]. The approach of [22] can only be applied to so-called *strongly-silent* protocols. However, this class does not contain many fast and succinct protocols recently developed for different tasks [4,17,20].

We are able to verify all six protocols reported in [22]. Further, we are also able to verify the fast Majority [5] protocol as well as the succinct protocols Flock-of-birds [20, Sect. 3] and Threshold [20]. All three protocols are not strongly-silent. Although our approach is more general and complete, the time to verify many strongly-silent protocol does not differ significantly between the two approaches. Exceptions are the Flock-of-birds [28] protocols where we are faster ([22] reaches the timeout at $c = 55$) as well as the Remainder and the Flock-of-birds-threshold-$n$ protocols where we are substantially slower ([22] reaches the timeout at $m = 80$ and $c = 350$, respectively). Loosely speaking, the approach of [22] can be faster because they compute inductive overapproximations using an iterative procedure instead of *PotReach*. In some instances already a very weak overapproximation, much less precise than *PotReach*, suffices to verify the result. Our procedure can be adapted to accommodate this (it essentially amounts to first running the procedure of [22], and if it is inconclusive then run ours).

*Other Distributed Algorithms.* We have also used our approach to verify arbitrary LTL liveness properties of non-parameterized systems with arbitrary communication structure. For this we apply standard automata-theoretic techniques and

[5] Notice that for each protocol we check correctness for all inputs; we cannot yet automatically verify that infinitely many protocols are correct, each of them for all possible inputs.

construct a product of the system and a *limit-deterministic Büchi automaton* for the negation of the property. Checking that no fair runs of the product are accepted by the automaton reduces to checking a stable termination property.

Since we only check correctness of one single finite-state system, we can also apply a probabilistic model checker based on state-space exploration. However, our technique delivers a stage graph, which plays two roles. First, it gives an explanation of why the property holds in terms of invariants and ranking functions, and second, it is a certificate of correctness that can be efficiently checked by independent means.

We verify liveness properties for several leader election and mutex algorithms from the literature [3,40,42,44,50,59,61,64] under the assumption of a probabilistic scheduler. For the leader election algorithms, we check that a leader is eventually chosen; for the mutex algorithms, we check that the first process enters its critical section infinitely often.

*Comparison with PRISM* [49]. We compared execution times for verification by our technique and by PRISM on the same models. While PRISM only needs a few seconds to verify instances of the mutex algorithms [3,40,50,59,61,64] where we reach the time limit, it reaches the memory limit for the two leader election algorithms [42,44] already for 70 and 71 processes, which we can still verify.

## 9 Conclusion and Further Work

We have presented stage graphs, a sound and complete technique for the verification of stable termination properties of replicated systems, an important class of parameterized systems. Using deep results of the theory of Petri nets, we have shown that Presburger stage graphs, a class of stage graphs whose correctness can be reduced to the satisfiability problem of Presburger arithmetic, are also sound and complete. This provides a decision procedure for the verification of termination properties, which is of theoretical nature since it involves a blind enumeration of candidates for Presburger stage graphs. For this reason, we have presented a technique for the algorithmic construction of Presburger stage graphs, designed to exploit the strengths of SMT-solvers for existential Presburger formulas, i.e., integer linear constraints. Loosely speaking, the technique searches for *linear* functions certifying the progress between stages, even though only the much larger class of Presburger functions guarantees completeness.

We have conducted extensive experiments on a large set of benchmarks. In particular, our approach is able to prove correctness of nearly all the standard protocols described in the literature, including several protocols that could not be proved by the technique of [22], which only worked for so-called strongly-silent protocols. We have also successfully applied the technique to some self-stabilization algorithms, leader election and mutual exclusion algorithms.

Our technique is based on the mechanized search for invariants and ranking functions. It avoids the use of state-space exploration as much as possible. For this reason, it also makes sense as a technique for the verification of liveness properties of non-parameterized systems with a finite but very large state space.

## References

1. Abdulla, P.A.: Regular model checking. Int. J. Softw. Tools Technol. Transf. **14**(2), 109–118 (2012). https://doi.org/10.1007/s10009-011-0216-8
2. Abdulla, P.A., Cerans, K., Jonsson, B., Tsay, Y.: General decidability theorems for infinite-state systems. In: Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS 1996, New Brunswick, New Jersey, USA, 27–30 July 1996, pp. 313–321. IEEE Computer Society (1996). https://doi.org/10.1109/LICS.1996.561359
3. Abdulla, P.A., Delzanno, G., Henda, N.B., Rezine, A.: Regular model checking without transducers (on efficient verification of parameterized systems). In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 721–736. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71209-1_56
4. Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. SIGACT News **49**(3), 63–73 (2018). https://doi.org/10.1145/3289137.3289150
5. Alistarh, D., Gelashvili, R., Vojnovic, M.: Fast and exact majority in population protocols. In: Georgiou, C., Spirakis, P.G. (eds.) Proceedings of the 34th ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, 21–23 July 2015, pp. 47–56. ACM (2015). https://doi.org/10.1145/2767386.2767429
6. Aminof, B., Rubin, S., Zuleger, F., Spegni, F.: Liveness of parameterized timed networks. In: Halldórsson, M.M., Iwama, K., Kobayashi, N., Speckmann, B. (eds.) ICALP 2015, Part II. LNCS, vol. 9135, pp. 375–387. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47666-6_30
7. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
8. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. In: Chaudhuri, S., Kutten, S. (eds.) Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, 25–28 July 2004, pp. 290–299. ACM (2004). https://doi.org/10.1145/1011767.1011810
9. Angluin, D., Aspnes, J., Diamadi, Z., Fischer, M.J., Peralta, R.: Computation in networks of passively mobile finite-state sensors. Distrib. Comput. **18**(4), 235–253 (2006). https://doi.org/10.1007/s00446-005-0138-3
10. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007). https://doi.org/10.1007/s00446-007-0040-2
11. Athanasiou, K., Liu, P., Wahl, T.: Unbounded-thread program verification using thread-state equations. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 516–531. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_35
12. Babiak, T., et al.: The Hanoi omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015, Part I. LNCS, vol. 9206, pp. 479–486. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_31
13. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
14. Basler, G., Mazzucchi, M., Wahl, T., Kroening, D.: Symbolic counter abstraction for concurrent software. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 64–78. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_9

15. Berman, L.: The complexitiy of logical theories. Theoret. Comput. Sci. **11**, 71–77 (1980). https://doi.org/10.1016/0304-3975(80)90037-7
16. Bloem, R., Jacobs, S., Khalimov, A., Konnov, I., Rubin, S., Veith, H., Widder, J.: Decidability of Parameterized Verification. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers (2015). https://doi.org/10.2200/S00658ED1V01Y201508DCT013
17. Blondin, M., Esparza, J., Genest, B., Helfrich, M., Jaax, S.: Succinct population protocols for presburger arithmetic. In: Proceedings of 37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, 10–13 March 2020, Montpellier, France. LIPIcs, vol. 154, pp. 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). https://doi.org/10.4230/LIPIcs.STACS.2020.40
18. Blondin, M., Esparza, J., Helfrich, M., Kučera, A., Meyer, P.J.: Artifact evaluation VM and instructions to generate experimental results for the CAV20 paper: checking Qualitative Liveness Properties of Replicated Systems with Stochastic Scheduling. figshare:12295982 (2020). https://doi.org/10.6084/m9.figshare.12295982.v2
19. Blondin, M., Esparza, J., Helfrich, M., Kučera, A., Meyer, P.J.: Checking qualitative liveness properties of replicated systems with stochastic scheduling. arXiv:2005.03555 [cs.LO] (2020). https://arxiv.org/abs/2005.03555
20. Blondin, M., Esparza, J., Jaax, S.: Large flocks of small birds: on the minimal size of population protocols. In: Proceedings of 35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, 28 February - 3 March 2018, Caen, France. LIPIcs, vol. 96, pp. 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.STACS.2018.16
21. Blondin, M., Esparza, J., Jaax, S.: Peregrine: a tool for the analysis of population protocols. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018, Part I. LNCS, vol. 10981, pp. 604–611. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_34
22. Blondin, M., Esparza, J., Jaax, S., Meyer, P.J.: Towards efficient verification of population protocols. In: Schiller, E.M., Schwarzmann, A.A. (eds.) Proceedings of 36th ACM Symposium on Principles of Distributed Computing, PODC 2017, Washington, DC, USA, 25–27 July 2017, pp. 423–430. ACM (2017). https://doi.org/10.1145/3087801.3087816
23. Blondin, M., Esparza, J., Kučera, A.: Automatic analysis of expected termination time for population protocols. In: Schewe, S., Zhang, L. (eds.) Proceedings of 29th International Conference on Concurrency Theory, CONCUR 2018, 4–7 September 2018, Beijing, China. LIPIcs, vol. 118, pp. 33:1–33:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018). https://doi.org/10.4230/LIPIcs.CONCUR.2018.33
24. Blondin, M., Finkel, A., Haase, C., Haddad, S.: The logical view on continuous petri nets. ACM Trans. Comput. Log. (TOCL) **18**(3), 24:1–24:28 (2017). https://doi.org/10.1145/3105908
25. Bouajjani, A., Jonsson, B., Nilsson, M., Touili, T.: Regular model checking. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 403–418. Springer, Heidelberg (2000). https://doi.org/10.1007/10722167_31
26. Browne, M.C., Clarke, E.M., Grumberg, O.: Reasoning about networks with many identical finite state processes. Inf. Comput. **81**(1), 13–31 (1989). https://doi.org/10.1016/0890-5401(89)90026-6
27. Cardelli, L., Csikász-Nagy, A.: The cell cycle switch computes approximate majority. Sci. Rep. **2**(1), 656 (2012). https://doi.org/10.1038/srep00656
pagebreak

28. Chatzigiannakis, I., Michail, O., Spirakis, P.G.: Algorithmic verification of population protocols. In: Dolev, S., Cobb, J., Fischer, M., Yung, M. (eds.) SSS 2010. LNCS, vol. 6366, pp. 221–235. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16023-3_19
29. Chen, Y., Hong, C., Lin, A.W., Rümmer, P.: Learning to prove safety over parameterised concurrent systems. In: Stewart, D., Weissenbacher, G. (eds.) Proceedings of 17th International Conference on Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, 2–6 October 2017, pp. 76–83. IEEE (2017). https://doi.org/10.23919/FMCAD.2017.8102244
30. Clarke, E., Talupur, M., Touili, T., Veith, H.: Verification by network decomposition. In: Gardner, P., Yoshida, N. (eds.) CONCUR 2004. LNCS, vol. 3170, pp. 276–291. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_18
31. Clément, J., Delporte-Gallet, C., Fauconnier, H., Sighireanu, M.: Guidelines for the verification of population protocols. In: Proceedings of 31st International Conference on Distributed Computing Systems, ICDCS 2011, Minneapolis, Minnesota, USA, 20–24 June 2011, pp. 215–224. IEEE Computer Society (2011). https://doi.org/10.1109/ICDCS.2011.36
32. Cooper, D.C.: Theorem proving in arithmetic without multiplication. Mach. Intell. **7**, 91–99 (1972)
33. Czerwinski, W., Lasota, S., Lazic, R., Leroux, J., Mazowiecki, F.: The reachability problem for petri nets is not elementary. In: Charikar, M., Cohen, E. (eds.) Proceedings of 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, 23–26 June 2019, pp. 24–33. ACM (2019). https://doi.org/10.1145/3313276.3316369
34. Emerson, E.A., Namjoshi, K.S.: On reasoning about rings. Int. J. Found. Comput. Sci. **14**(4), 527–550 (2003). https://doi.org/10.1142/S0129054103001881
35. Esparza, J., Ganty, P., Leroux, J., Majumdar, R.: Model checking population protocols. In: Lal, A., Akshay, S., Saurabh, S., Sen, S. (eds.) Proceedings of 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, Chennai, India, 13–15 December 2016. LIPIcs, vol. 65, pp. 27:1–27:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.FSTTCS.2016.27
36. Esparza, J., Ganty, P., Leroux, J., Majumdar, R.: Verification of population protocols. Acta Inf. **54**(2), 191–215 (2017). https://doi.org/10.1007/s00236-016-0272-3
37. Esparza, J., Ledesma-Garza, R., Majumdar, R., Meyer, P., Niksic, F.: An SMT-based approach to coverability analysis. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 603–619. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08867-9_40
38. Esparza, J., Meyer, P.J.: An SMT-based approach to fair termination analysis. In: Kaivola, R., Wahl, T. (eds.) Proceedings of 15th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, 27–30 September 2015, pp. 49–56. IEEE (2015)
39. Finkel, A., Schnoebelen, P.: Well-structured transition systems everywhere!. Theoret. Comput. Sci. **256**(1–2), 63–92 (2001). https://doi.org/10.1016/S0304-3975(00)00102-X

40. Fribourg, L., Olsén, H.: Reachability sets of parameterized rings as regular languages. In: Moller, F. (ed.) Proceedings of 2nd International Workshop on Verification of Infinite State Systems, Infinity 1997, Bologna, Italy, 11–12 July 1997. Electronic Notes in Theoretical Computer Science, vol. 9, p. 40. Elsevier (1997). https://doi.org/10.1016/S1571-0661(05)80427-X
41. German, S.M., Sistla, A.P.: Reasoning about systems with many processes. J. ACM **39**(3), 675–735 (1992). https://doi.org/10.1145/146637.146681
42. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990). https://doi.org/10.1016/0020-0190(90)90107-9
43. Hopcroft, J.E., Pansiot, J.: On the reachability problem for 5-dimensional vector addition systems. Theoret. Comput. Sci. **8**, 135–159 (1979). https://doi.org/10.1016/0304-3975(79)90041-0
44. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizing mutual exclusion. In: Dwork, C. (ed.) Proceedings of 9th Annual ACM Symposium on Principles of Distributed Computing, PODC 1990, Quebec City, Quebec, Canada, 22–24 August 1990, pp. 119–131. ACM (1990). https://doi.org/10.1145/93385.93409
45. Jancar, P., Purser, D.: Structural liveness of petri nets is expspace-hard and decidable. Acta Inf. **56**(6), 537–552 (2019). https://doi.org/10.1007/s00236-019-00338-6
46. Kaiser, A., Kroening, D., Wahl, T.: Dynamic cutoff detection in parameterized concurrent programs. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 645–659. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14295-6_55
47. Kaiser, A., Kroening, D., Wahl, T.: A widening approach to multithreaded program verification. ACM Trans. Program. Lang. Syst. **36**(4), 14:1–14:29 (2014). https://doi.org/10.1145/2629608
48. Křetínský, J., Meggendorfer, T., Sickert, S.: Owl: a library for $\omega$-words, automata, and LTL. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 543–550. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_34
49. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47
50. Lehmann, D., Rabin, M.O.: On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In: White, J., Lipton, R.J., Goldberg, P.C. (eds.) Proceedings of 8th Annual ACM Symposium on Principles of Programming Languages, POPL 1981, Williamsburg, Virginia, USA, January 1981, pp. 133–138. ACM Press (1981). https://doi.org/10.1145/567532.567547
51. Lengál, O., Lin, A.W., Majumdar, R., Rümmer, P.: Fair termination for parameterized probabilistic concurrent systems. In: Legay, A., Margaria, T. (eds.) TACAS 2017, Part I. LNCS, vol. 10205, pp. 499–517. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54577-5_29
52. Leroux, J.: Vector addition systems reachability problem (a simpler solution). In: Voronkov, A. (ed.) Proceedings of the Alan Turing Centenary Conference, Turing 100, Manchester, UK, 22–25 June 2012. EPiC Series in Computing, vol. 10, pp. 214–228. EasyChair (2012). https://doi.org/10.29007/bnx2
53. Leroux, J.: Presburger vector addition systems. In: Proceedings of 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, 25–28 June 2013. pp. 23–32. IEEE Computer Society (2013). https://doi.org/10.1109/LICS.2013.7

54. Leroux, J.: Vector addition system reversible reachability problem. Log. Methods Comput. Sci. **9**(1) (2013). https://doi.org/10.2168/LMCS-9(1:5)2013
55. Lin, A.W., Rümmer, P.: Liveness of randomised parameterised systems under arbitrary schedulers. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016, Part II. LNCS, vol. 9780, pp. 112–133. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41540-6_7
56. Moran, P.A.P.: Random processes in genetics. Math. Proc. Cambridge Philos. Soc. **54**(1), 60–71 (1958). https://doi.org/10.1017/S0305004100033193
57. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
58. Navlakha, S., Bar-Joseph, Z.: Distributed information processing in biological and computational systems. Commun. ACM **58**(1), 94–102 (2015). https://doi.org/10.1145/2678280
59. Nilsson, M.: Regular model checking. Ph.D. thesis, Uppsala University (2000)
60. Pang, J., Luo, Z., Deng, Y.: On automatic verification of self-stabilizing population protocols. In: Proceedings of 2nd IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering, TASE 2008, 17–19 June 2008, Nanjing, China, pp. 185–192. IEEE Computer Society (2008). https://doi.org/10.1109/TASE.2008.8
61. Peterson, G.L.: Myths about the mutual exclusion problem. Inf. Process. Lett. **12**(3), 115–116 (1981). https://doi.org/10.1016/0020-0190(81)90106-X
62. Presburger, M.: Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. Comptes Rendus du I$^{er}$ Congrès des mathématiciens des pays slaves, pp. 192–201 (1929)
63. Sun, J., Liu, Y., Dong, J.S., Pang, J.: PAT: towards flexible verification under fairness. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 709–714. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_59
64. Szymanski, B.K.: A simple solution to Lamport's concurrent programming problem with linear wait. In: Lenfant, J. (ed.) Proceedings of 2nd International Conference on Supercomputing, ICS 1988, Saint Malo, France, 4–8 July 1988, pp. 621–626. ACM (1988). https://doi.org/10.1145/55364.55425
65. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proceedings of 26th Annual Symposium on Foundations of Computer Science, FOCS 1985, Portland, Oregon, USA, 21–23 October 1985, pp. 327–338. IEEE Computer Society (1985). https://doi.org/10.1109/SFCS.1985.12

# B Peregrine 2.0: Explaining Correctness of Population Protocols through Stage Graphs

This chapter has been published as a **peer-reviewed conference paper**.

> Javier Esparza, Martin Helfrich, Stefan Jaax, and Philipp J. Meyer. "Peregrine 2.0: Explaining Correctness of Population Protocols Through Stage Graphs". In: *Automated Technology for Verification and Analysis*. Ed. by Dang Van Hung and Oleg Sokolsky. Cham: Springer International Publishing, 2020, pp. 550–556. DOI: 10.1007/978-3-030-59152-6_32

**Summary.** We extend the tool Peregrine with an automatic verification procedure that uses the efficient stage graph construction of [BEH+20]. We give a high-level explanation of stage graphs and present a new visualization technique for stage graphs. This graphical Venn-diagram representation helps users to understand the computation of population protocols. Additionally, we highlight new features related to stage graphs, such as the visualization of simulations inside of stage graphs, the automatic analysis of the speed of a protocol, and the detection of bugs through counterexamples. We demonstrate how the new functionality helps users to understand population protocols with multiple examples.

**Contributions of thesis author.** The author played a pivotal role in the composition and revision of the manuscript. They actively participated in joint discussions and contributed significantly to the creation of techniques presented in the paper. Noteworthy individual contributions include the implementation and integration of the efficient verification technique, the automatic speed analysis, as well as the initial idea for the visualization of stage graphs and simulations.

# Peregrine 2.0: Explaining Correctness of Population Protocols Through Stage Graphs

Javier Esparza, Martin Helfrich, Stefan Jaax, and Philipp J. Meyer(✉)

Technical University of Munich, Munich, Germany
{esparza,helfrich,jaax,meyerphi}@in.tum.de

**Abstract.** We present a new version of Peregrine, the tool for the analysis and parameterized verification of population protocols introduced in [Blondin et al., CAV'2018]. Population protocols are a model of computation, intensely studied by the distributed computing community, in which mobile anonymous agents interact stochastically to perform a task. Peregrine 2.0 features a novel verification engine based on the construction of stage graphs. Stage graphs are proof certificates, introduced in [Blondin et al., CAV'2020], that are typically succinct and can be independently checked. Moreover, unlike the techniques of Peregrine 1.0, the stage graph methodology can verify protocols whose executions never terminate, a class including recent fast majority protocols. Peregrine 2.0 also features a novel proof visualization component that allows the user to interactively explore the stage graph generated for a given protocol.

**Keywords:** Population protocols · Distributed computing · Parameterized verification · Stage graphs.

## 1 Introduction

We present Peregrine 2.0[1], a tool for analysis and parameterized verification of population protocols. Population protocols are a model of computation, intensely studied by the distributed computing community, in which an arbitrary number of indistinguishable agents interact stochastically in order to decide a given property of their initial configuration. For example, agents could initially be in one of two possible states, "yes" and "no", and their task could consist of deciding whether the initial configuration has a majority of "yes" agents or not.

Verifying correctness and/or efficiency of a protocol is a very hard problem, because the semantics of a protocol is an infinite collection of finite-state Markov

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 787367 (PaVeS). We thank Michael Blondin for contributions to the frontend and Philip Offtermatt for improvements of the simulation backend.

[1] Peregrine 2.0 is available at https://peregrine.model.in.tum.de/.

chains, one for each possible initial configuration. Peregrine 1.0 [5] was the first tool for the automatic verification of population protocols. It relies on theory developed in [6], and is implemented on top of the Z3 SMT-solver.

Peregrine 1.0 could only verify protocols whose agents eventually never change their state (and not only their answer). This constraint has become increasingly restrictive, because it is not satisfied by many efficient and succinct protocols recently developed for different tasks [1,2,4]. Further, Peregrine 1.0 was unable to provide correctness certificates and the user had to trust the tool. Finally, Peregrine 1.0 did not provide any support for computing parameterized bounds on the expected number of interactions needed to reach a stable consensus, i.e., bounds like "$\mathcal{O}(n^2 \log n)$ interactions, where $n$ is the number of agents".

Peregrine 2.0 addresses these three issues. It features a novel verification engine based on theory developed in [3,7], which, given a protocol and a task description, attempts to construct a stage graph. Stage graphs are proof certificates that can be checked by independent means, and not only prove the protocol correct, but also provide a bound on its expected time-to-consensus. Stages represent milestones reached by the protocol on the way to consensus. Stage graphs are usually small, and help designers to understand why a protocol works. The second main novel feature of Peregrine 2.0 is a visualization component that offers a graphical and explorable representation of the stage graph.

The paper is organized as follows. Section 2 introduces population protocols and sketches the correctness proof of a running example. Section 3 describes the stage graph generated for the example by Peregrine 2.0, and shows that it closely matches the human proof. Section 4 describes the visualization component.

## 2 Population Protocols

A *population protocol* consists of a set $Q$ of *states* with a subset $I \subseteq Q$ of *initial states*, a set $T \subseteq Q^2 \times Q^2$ of *transitions*, and an *output function* $O : Q \to \{0, 1\}$ assigning to each state a boolean output. Intuitively, a transition $q_1, q_2 \mapsto q_3, q_4$ means that two agents in states $q_1, q_2$ can interact and simultaneously move to states $q_3, q_4$. A *configuration* is a mapping $C : Q \to \mathbb{N}$, where $C(q)$ represents the number of agents in a state $q$. An *initial configuration* is a mapping $C : I \to \mathbb{N}$. A configuration has *consensusb* $\in \{0, 1\}$ if all agents are in states with output $b$. We write configurations using a set-like notation, e.g. $C = \wr \mathtt{y}, \mathtt{n}, \mathtt{n} \wr$ or $C = \wr \mathtt{y}, 2 \cdot \mathtt{n} \wr$ is the configuration where $C(\mathtt{y}) = 1$, $C(\mathtt{n}) = 2$ and $C(q) = 0$ for $q \notin \{\mathtt{y}, \mathtt{n}\}$.

*Running Example: Majority Voting.* The goal of this protocol is to conduct a vote by majority in a distributed way. The states are $\{\mathtt{Y}, \mathtt{N}, \mathtt{y}, \mathtt{n}\}$. Initially, all agents are in state $\mathtt{Y}$ or $\mathtt{N}$, according to how they vote. The goal of the protocol is that the agents determine whether at least 50% of them vote "yes".

The output function is $O(\mathtt{Y}) = O(\mathtt{y}) = 1$ and $O(\mathtt{N}) = O(\mathtt{n}) = 0$. When two agents interact, they change their state according to the following transitions:

$$a : \mathtt{Y\,N} \mapsto \mathtt{y\,n} \qquad b : \mathtt{Y\,n} \mapsto \mathtt{Y\,y} \qquad c : \mathtt{N\,y} \mapsto \mathtt{N\,n} \qquad d : \mathtt{y\,n} \mapsto \mathtt{y\,y}$$

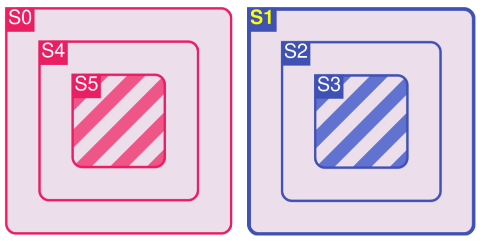
Intuitively, agents are either active ($\mathtt{Y}$, $\mathtt{N}$) or passive ($\mathtt{y}$, $\mathtt{n}$). By transition $a$, when active agents with opposite opinions meet, they become passive. Transitions $b$ and $c$ let active agents change the opinion of passive agents. Transition $d$ handles the case of a tie.

*Computations in Population Protocols.* Computations use a stochastic model: starting from an initial configuration $C_0$, two agents are repeatedly picked, uniformly at random, and the corresponding transition is applied. This gives rise to an infinite sequence $C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \ldots$ of configurations, called a *run*. A run *stabilizes* to consensus $b \in \{0, 1\}$ if from some point on all configurations have consensus $b$. Intuitively, in a run that stabilizes to $b$ the agents eventually agree on the answer $b$. Given a population protocol $\mathcal{P}$ and a *predicate* $\varphi$ that maps every configuration $C$ to a value in $\{0, 1\}$, we say that $\mathcal{P}$ *computes* $\varphi$ if for every initial configuration $C$, a run starting at $C$ stabilizes to consensus $\varphi(C)$ with probability 1. The *correctness problem* consists of deciding, given $\mathcal{P}$ and $\varphi$, whether $\mathcal{P}$ computes $\varphi$. Intuitively, a correct protocol almost surely converges to the consensus specified by the predicate. Majority Voting is correct and computes the predicate that assigns 1 to the configurations where initially at least 50% of the agents are in state $\mathtt{Y}$, i.e. we have $\varphi(C) = (C(\mathtt{Y}) \geq C(\mathtt{N}))$.

*Majority Voting is Correct.* To intuitively understand why the protocol is correct, it is useful to split a run into *phases*. The first phase starts in the initial configuration, and ends when two agents interact using transition $a$ for the last time. Observe that this moment arrives with probability 1 because passive agents can never become active again. Further, at the end of the first phase either all active agents are in state $\mathtt{Y}$, or they are all in state $\mathtt{N}$. The second phase ends when the agents reach a consensus for the first time, that is, the first time that either all agents are in states $\mathtt{Y}, \mathtt{y}$, or all are in states $\mathtt{N}, \mathtt{n}$. To see that the second phase ends with probability 1, consider three cases. If initially there is a majority of "yes", then at the end of the first phase no agent is in state $\mathtt{N}$, and at least one is in state $\mathtt{Y}$. This agent eventually moves all passive agents in state $\mathtt{n}$ to state $\mathtt{y}$ using transition $b$, reaching a "yes" consensus. The case with an initial majority of "no" is symmetric. If initially there is a tie, then at the end of the first phase all agents are passive, and transition $d$ eventually moves all agents in state $\mathtt{n}$ to $\mathtt{y}$, again resulting in a "yes" consensus. The third phase is the rest of the run. We observe that once the agents reach a consensus no transition is enabled, and so the agents remain in this consensus, proving that the protocol is correct.

## 3 Protocol Verification with Peregrine 2.0

Peregrine 2.0 allows the user to specify and edit population protocols. (Our running example is listed in the distribution as *Majority Voting*.) After choosing a protocol, the user can simulate it and gather statistics, as in Peregrine 1.0 [5]. The main feature of Peregrine 2.0 is its new verification engine based on stage graphs, which closely matches the "phase-reasoning" of the previous section.



| Stage | Constraint | Certificate | Speed |
|---|---|---|---|
| $S_0$ | $\mathcal{R}$ | $C(\mathtt{Y})$ | $\mathcal{O}(n^2 \log n)$ |
| $S_4$ | $\mathcal{R} \wedge C(\mathtt{Y}) = 0$ | $C(\mathtt{y})$ | $2^{\mathcal{O}(n \log n)}$ |
| $S_5$ | $\mathcal{R} \wedge C(\mathtt{Y}){+}C(\mathtt{y}) = 0$ | $\perp$ | $\perp$ |
| $S_1$ | $\mathcal{R}'$ | $C(\mathtt{N})$ | $\mathcal{O}(n^2 \log n)$ |
| $S_2$ | $\mathcal{R}' \wedge C(\mathtt{N}) = 0$ | $C(\mathtt{n})$ | $\mathcal{O}(n^2 \log n)$ |
| $S_3$ | $\mathcal{R}' \wedge C(\mathtt{N}){+}C(\mathtt{n}) = 0$ | $\perp$ | $\perp$ |

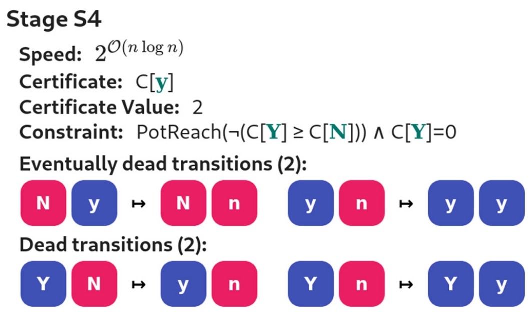
**Fig. 1.** Stage graphs for Majority Voting protocol with constraints, certificates and speeds. The expression $\mathcal{R}$ and $\mathcal{R}'$ denote abstractions of the reachability relation, which are a bit long and therefore omitted for clarity.

*Stage Graphs.* A *stage graph* is a directed acyclic graph whose nodes, called *stages*, are possibly infinite sets of configurations, finitely described by a Presburger formula. Stages are *inductive*, i.e. closed under reachability. There is an edge $S \to S'$ to a *child* stage $S'$ if $S' \subset S$, and no other stage $S''$ satisfies $S' \subset S'' \subset S$. Peregrine 2.0 represents stage graphs as Venn diagrams like the ones on the left of Fig. 1. Stages containing no other stages are called *terminal*, and otherwise *non-terminal*. Intuitively, a phase starts when a run enters a stage, and ends when it reaches one of its children.

Each non-terminal stage $S$ comes equipped with a *certificate*. Intuitively, a certificate proves that runs starting at any configuration of $S$ will almost surely reach one of its children and, since $S$ is inductive, get trapped there forever. Loosely speaking, certificates take the form of ranking functions bounding the distance of a configuration to the children of $S$, and are also finitely represented by Presburger formulas. Given a configuration $C$ and a certificate $f$, runs starting at $C$ reach a configuration $C'$ satisfying $f(C') < f(C)$ with probability 1.

To verify that a protocol computes a predicate $\varphi$ we need two stage graphs, one for each output. The roots of the first stage graph contain all initial configurations $C$ with $\varphi(C) = 0$ and the terminal stages contain only configurations with consensus 0. The second handles the case when $\varphi(C) = 1$.

*Stage Graphs for Majority Voting.* For the Majority Voting protocol Peregrine 2.0 generates the two stage graphs of Fig. 1 in a completely automatic way. By clicking on a stage, say $S_4$, the information shown in Fig. 2 is displayed. The constraint describes the set of configurations of the stage (Fig. 1 shows the constraints for all stages). In particular, all the configurations of $S_4$ satisfy $C(\mathtt{Y}) = 0$, that is, all agents initially in state Y have already become passive. The certificate indicates that a run starting at a configuration $C \in S_4 \setminus S_5$ eventually reaches $S_5$ or a configuration $C' \in S_4 \setminus S_5$ such that $C'(\mathtt{y}) < C(\mathtt{y})$. Peregrine 2.0 also displays a list of *dead transitions* that can never occur again from any configuration of $S_4$, and a list of *eventually dead transitions*, which will become dead whenever a child stage, in this case $S_5$, is reached.


**Stage S4**

**Speed:** $2^{\mathcal{O}(n \log n)}$
**Certificate:** C[y]
**Certificate Value:** 2
**Constraint:** PotReach(¬(C[Y] ≥ C[N])) ∧ C[Y]=0

**Eventually dead transitions (2):**

N y ↦ N n    y n ↦ y y

**Dead transitions (2):**

Y N ↦ y n    Y n ↦ Y y

**Fig. 2.** Details of stage $S_4$ in Fig. 1 at configuration $\{\!\!\{\mathtt{N}, 4 \cdot \mathtt{n}, 2 \cdot \mathtt{y}\}\!\!\}$. The terms C[q] are the number of agents $C(q)$ in state $q$.

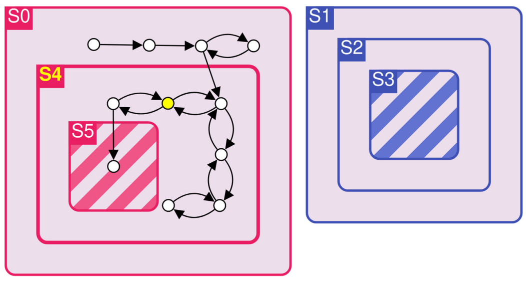


**Fig. 3.** Partially constructed Markov chain after a simulation of the Majority Voting protocol inside the protocol's stage graphs, with ○ $= \{\!\!\{\mathtt{N}, 4 \cdot \mathtt{n}, 2 \cdot \mathtt{y}\}\!\!\}$ selected. (Color figure online)

While they are automatically generated, these stage graphs closely map the intuition above. The three stages of each graph naturally correspond to the three phases of the protocol: $S_0$ and $S_1$ correspond to the first phase (we reduce $C(\mathtt{Y})$ or $C(\mathtt{N})$), $S_2$ and $S_4$ to the second phase ($C(\mathtt{Y})$ or $C(\mathtt{N})$ is zero, and we reduce $C(\mathtt{y})$ or $C(\mathtt{n})$), and $S_3$ and $S_5$ to the third phase (all agents are in consensus).

*Speed.* Because agents interact randomly, the length of the phase associated to a stage is a random variable (more precisely, a variable for each number of agents). The expected value of this variable is called the *speed* of the stage. A stage has speed $\mathcal{O}(f(n))$ if for every $n$ the expected length of the phase for configurations with $n$ agents is at most $c \cdot f(n)$ for some constant $c$. Peregrine 2.0 computes an upper bound for the speed of a stage using the techniques of [7]. The last column of Fig. 1 gives the upper bounds on the speed of all stages. Currently, Peregrine 2.0 can prove one of the bounds $\mathcal{O}(n^2 \log n)$, $\mathcal{O}(n^3)$, $\mathcal{O}(n^k)$ for some $k$ and $2^{\mathcal{O}(n \log n)}$. Observe that for stage $S_4$ of Majority Voting the tool returns $2^{\mathcal{O}(n \log n)}$. Majority Voting is indeed very inefficient, much faster protocols exist.

## 4 Visualizing Runs in the Stage Graph

To further understand the protocol, Peregrine 2.0 allows the user to simulate a run and monitor its progress through the stage graph. The simulation is started at a chosen initial configuration or a precomputed example configuration of a stage. The current configuration is explicitly shown and also highlighted as a yellow circle in the stage graph. To choose the next pair of interacting agents, the user can click on them. The resulting interaction is visualized, and the successor configuration is automatically placed in the correct stage, connected to the previous configuration. After multiple steps, this partially constructs the underlying Markov chain of the system as shown in Fig. 3. One can also navigate the current run by clicking on displayed configurations or using the PREV and NEXT buttons.

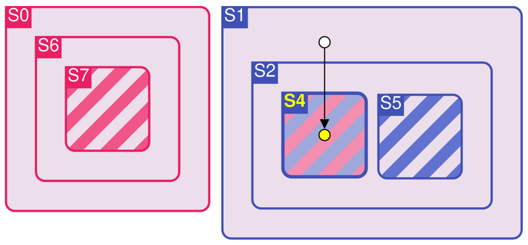


**Fig. 4.** Counterexample automatically found by Peregrine when verifying Majority Voting (broken), shown in the stage graphs as a run from ○ = $\wr$Y, N$\int$ to ● = $\wr$y, n$\int$. The graph with root $S_1$ is only a partial stage graph, because stage $S_4$ contains configurations that do not have the correct consensus.

Beyond choosing pairs of agents one by one, the user can simulate a full run of the protocol by clicking on PLAY. The acceleration slider allows to speed up this simulation. However, if the overall speed of the protocol is very slow, a random run might not make progress in a reasonable time frame. An example for this is the Majority Voting protocol for populations with a small majority for N, where the expected number of interactions to go from $S_4$ to $S_5$ is $2^{\mathcal{O}(n \log n)}$. Thus, even for relatively small configurations like $\wr 4 \cdot \mathtt{Y}, 5 \cdot \mathtt{N} \int$ a random run is infeasible. To make progress in these cases, one can click on PROGRESS. This automatically chooses a transition that reduces the value of the certificate. Intuitively, reducing the certificate's value guides the run towards a child stage and thus, the run from $S_4$ to $S_5$ needs at most $n$ steps. To visualize the progress, the value of the stage's certificate for the current configuration is displayed in the stage details as in Fig. 2 and next to the PROGRESS button.

*Finding Counterexamples.* The speed of stage $S_4$ with certificate $C(\mathtt{y})$ is so low because of transition $d : \mathtt{y}\,\mathtt{n} \mapsto \mathtt{y}\,\mathtt{y}$ that increases the value of the certificate and may be chosen with high probability. Removing the transition $d$ makes the protocol faster (this variant is listed in the distribution as "Majority Voting (broken)"). However, then Peregrine cannot verify the protocol anymore, and it even finds a counterexample: a run that does not stabilize to the correct consensus. Figure 4 shows the counterexample ending in the configuration $\wr \mathtt{y}, \mathtt{n} \int$ from the initial configuration $\wr \mathtt{Y}, \mathtt{N} \int$, i.e. a configuration with a tie. In this case, the configuration should stabilize to 1, but no transition is applicable at $\wr \mathtt{y}, \mathtt{n} \int$, which does not have consensus 1. This clearly shows why we need the transition $d$. Note however that the left part with root stage $S_0$ in Fig. 4 is a valid stage graph, so the modified protocol works correctly in the ,negative case. This helps locate the cause of the problem.

## References

1. Alistarh, D., Gelashvili, R.: Recent algorithmic advances in population protocols. SIGACT News **49**(3), 63–73 (2018)
2. Blondin, M., Esparza, J., Genest, B., Helfrich, M., Jaax, S.: Succinct population protocols for Presburger arithmetic. In: STACS. LIPIcs, vol. 154, pp. 40:1–40:15 (2020). https://doi.org/10.4230/LIPIcs.STACS.2020.40
3. Blondin, M., Esparza, J., Helfrich, M., Kučera, A., Meyer, P.J.: Checking qualitative liveness properties of replicated systems with stochastic scheduling. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12225, pp. 372–397. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53291-8_20
4. Blondin, M., Esparza, J., Jaax, S.: Large flocks of small birds: on the minimal size of population protocols. In: STACS. LIPIcs, vol. 96, pp. 16:1–16:14 (2018)
5. Blondin, M., Esparza, J., Jaax, S.: PEREGRINE: a tool for the analysis of population protocols. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 604–611. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_34
6. Blondin, M., Esparza, J., Jaax, S., Meyer, P.J.: Towards efficient verification of population protocols. In: PODC, pp. 423–430. ACM (2017)
7. Blondin, M., Esparza, J., Kučera, A.: Automatic analysis of expected termination time for population protocols. In: CONCUR. LIPIcs, pp. 33:1–33:16 (2018)

# C Abstraction-Based Segmental Simulation of Chemical Reaction Networks

This chapter has been published as a **peer-reviewed conference paper**.

Martin Helfrich, Milan Češka, Jan Křetínský, and Štefan Martiček. “Abstraction-Based Segmental Simulation of Chemical Reaction Networks”. In: *Computational Methods in Systems Biology*. Ed. by Ion Petre and Andrei Păun. Cham: Springer International Publishing, 2022, pp. 41–60. DOI: 10.1007/978-3-031-15034-0_3

**Summary.** We introduce segmental simulation, a novel approximate simulation approach for chemical reaction networks based on memoization. This method accelerates the generation of new simulations by reusing segments from previous simulations. We explain how segmental simulation leverages a population-level abstraction to selectively reuse segments starting in the same region. Furthermore, we demonstrate that segmental simulation can be viewed as an abstraction technique with different levels of granularity. Through a detailed experimental evaluation on challenging benchmarks from the literature, we show that segmental simulation significantly speeds up the simulation process while preserving the dynamics of the original system. In comparison with other advanced simulation techniques, we establish that segmental simulation is competitive in terms of both efficiency and accuracy.

**Contributions of thesis author.** The author played a pivotal role in the composition and revision of the manuscript. They actively participated in joint discussions and contributed significantly to the development of the theoretical results presented in the paper. Noteworthy individual contributions include the initial idea for using memoization in simulation, the development of the segmental simulation technique, the introduction of the population-level abstraction, the inception of the lazy and adaptive segmental simulation methods, the implementation of the approach, as well as the experimental evaluation on benchmarks.

# Abstraction-Based Segmental Simulation of Chemical Reaction Networks

Martin Helfrich[1](✉), Milan Češka[2](✉), Jan Křetínský[1], and Štefan Martiček[2]

[1] Technical University of Munich, Munich, Germany
helfrich@in.tum.de
[2] Brno University of Technology, Brno, Czech Republic
ceskam@fit.vutbr.cz

**Abstract.** Simulating chemical reaction networks is often computationally demanding, in particular due to stiffness. We propose a novel simulation scheme where long runs are not simulated as a whole but assembled from shorter precomputed segments of simulation runs. On the one hand, this speeds up the simulation process to obtain multiple runs since we can reuse the segments. On the other hand, questions on diversity and genuineness of our runs arise. However, we ensure that we generate runs close to their true distribution by generating an appropriate abstraction of the original system and utilizing it in the simulation process. Interestingly, as a by-product, we also obtain a yet more efficient simulation scheme, yielding runs over the system's abstraction. These provide a very faithful approximation of concrete runs on the desired level of granularity, at a low cost. Our experiments demonstrate the speedups in the simulations while preserving key dynamical as well as quantitative properties.

**Keywords:** Chemical reaction networks · Population models · Stochastic simulation algorithm · Model abstraction

## 1 Introduction

**Chemical Reaction Networks (CRNs)** are a versatile language widely used for modeling and analysis of biochemical systems [10] as well as for high-level programing of molecular devices [6,34]. The time-evolution of CRNs is governed by the Chemical Master Equation that leads to a (potentially infinite) discrete-space, continuous-time Markov chain (CTMC) with "population" structure, describing how the probability of the molecular counts of each chemical species evolve in time. Many important biochemical systems feature complex dynamics, that are hard to analyze due to *state-space explosion, stochasticity, stiffness, and multimodality* of the population distributions [17,36]. This fundamentally limits the class of systems the existing techniques can handle effectively. There are several classes of

This work has been supported by the Czech Science Foundation grant GJ20-02328Y, the German Research Foundation (DFG) projects 378803395 (ConVeY) and 427755713 (GOPro) as well as the ERC Advanced Grant 787367 (PaVeS).

I. Petre and A. Păun (Eds.): CMSB 2022, LNBI 13447, pp. 41–60, 2022.
https://doi.org/10.1007/978-3-031-15034-0_3

approaches that try to circumvent these issues, in particular, (i) *stochastic simulation* avoids the explicit construction of the state space by sampling trajectories in the CRN, and (ii) *abstraction* builds a smaller/simpler model preserving the key dynamical properties and allowing for an efficient numerical analysis of the original CRN. Over the last two decades, there has been very active research on improving the performance and precision of these approaches, see the related work below. Yet, running thousands of simulations to approximate the stochastic behavior often takes many hours; and abstractions course enough to be analyzed easily often fail to capture the complex dynamics, e.g. oscillations in the notorious tiny (two-species) predator-prey system.

**Our Contribution.** In this paper, in several simple steps, we uniquely *combine* the simulation methods with the abstraction methods for CTMC states, further narrowing the performance gap. As the first step, we suggest leveraging memoization, a general optimization technique that pre-computes and stores partial results to speedup the consequent computation. In particular, when simulating from a current state we reuse previously generated pieces of runs, called *segments*, that start in "similar enough" states. Thus, rather than spending time on simulating a whole new run, we quickly stitch together the segments. To ensure a high variety of runs and generally a good correspondence to the original probability space of runs, we not only have to generate a sufficiently large number of segments; but it is crucial to also consider their length and the similarity of their starting states. We show how the latter two questions can be easily answered using the standard *interval abstraction* on the populations, yielding faithful yet fast simulations of the CTMC for the CRN.

In a second step, we also produce simulation runs over the (e.g. interval) abstraction of the CTMC, not only fast but also with low memory requirements, allowing for efficient analysis on the desired level of detail. To this end, we drop all the concrete information of each segment, keeping only its abstraction plus its *concrete end state*. This surprising choice of information allows us to *define transitions on the abstraction using the simulation* in a rather non-standard way. The resulting semantics and dynamics of the abstraction are non-Markovian, but capture the dynamics of the analyzed system very precisely. From the methodological perspective, the most interesting point is that simulation and abstraction can *help each other* although typically seen as disparate tools.

**Related Work.** To speed up the standard Stochastic Simulation Algorithm (SSA) [15], several *approximate multi-scale* simulation techniques have been proposed. They include advanced $\tau$-leaping methods [5,27], that use a Poisson approximation to adaptively take time steps leaping over many reactions assuming the system does not change significantly within these steps. Alternatively, various partitioning schemes for fast and slow reactions have been considered [4] allowing one to approximate the fast reactions by a quasi-steady-state assumption [17,32]. The idea of separating the slow and fast sub-networks has been further elaborated in hybrid simulations treating some appropriate species as continuous variables and the others as discrete ones [33]. As before, appropriate partitioning of the species is essential for the performance and accuracy, and thus several (adaptive) strategies have been proposed [14,24]. Recently a *deep learning* paradigm has been introduced to further shift the scalability of the CRN analysis. In [3], the authors learn from a set of stochastic simulations of the CRN a generator, in the form of a Generative Adversarial Network, that efficiently produces trajectories having a similar distribution as the trajectories in the original CRN. In [18] the authors go even further and learn from the simulations an estimator of the given statistic over the original CRN. The principal limitation of these approaches is the overhead related to the learning phase that typically requires a nontrivial number of the simulations of the original CRN.

To build a plausible and computationally tractable abstraction of CRNs, various *state-space reduction techniques* have been proposed that either truncate states of the underlying CTMC with insignificant probability [22,30,31] or leverage structural properties of the CTMC to aggregate/lump selected sets of states [1,2]. The *interval abstraction* of the species population is a widely used approach to mitigate the state-space explosion problem [13,29,37]. We define a segment of simulation runs as a sequence of transitions that can be seen as a single transition of the abstracted system. These "abstract" transitions in the interval abstractions are studied in [9] as "accelerated" transitions. Alternatively, several hybrid models have been considered levering a similar idea as the hybrid simulations. In [23], a pure deterministic semantic for large population species is used. The moment-based description for medium/high-copy number species was used in [19]. The LNA approximation and an adaptive partitioning of the species according to leap conditions (that is more general than partitioning based on population thresholds) were proposed in [7].

**Advantages of Our Approach.** We show that the proposed *segmental simulation scheme* preserves the key dynamical properties and its qualitative accuracy (with respect to the SSA baseline) is comparable with advanced simulation as well as deep-learning approaches. The scheme, however, provides a significant computational gain over these approaches. Consider a detailed analysis (including 100000 simulation runs) of the famous Toggle switch model reported in [24]. Using the $\tau$-leaping implementation in StochPY [28] is not feasible and the state-of-the-art adaptive hybrid simulation method requires a day to perform such an analysis. However, our approach needs less than two hours. Moreover, our lazy strategy does not require a computationally demanding pre-computation and typically significant benefits from reusing segments after a small number of simulations. This is the key advantage compared to the learning approaches [3,18], where a large number of simulations of the original CRN are required, as well as to approaches based on (approximate) bisimulation/lumping [11,26], requiring a complex analysis of the original model. The approaches based on hybrid formal analysis of the underlying CTMC [7,19,23] have to perform a computationally demanding analysis of conditioned stochastic processes. For example, in [7] the authors report that an analysis of the Viral infection model took more than 1 h. The segmental simulation using 10000 runs provides the same quantitative information in 3 min.

In our previous work [9], we proposed a semi-quantitative abstraction and analysis of CRNs focusing on explainability of the results and low computational complexity, however, providing only limited quantitative accuracy. The proposed simulation scheme provides significantly better accuracy as it keeps track of the current concrete state and thus avoids "jumps" to the abstract state's representative. This kind of rounding is a major source of error as exemplified in the predator-prey model, where the semi-quantitative abstraction of [9] failed to accurately preserve the oscillation and our new approach captures it faithfully.

## 2 Preliminaries

### Chemical Reaction Networks

*CRN Syntax.* A *chemical reaction network (CRN)* $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where $\Lambda$ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in $\Lambda$ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. $r_\tau$ and $p_\tau$ represent the stoichiometry of reactants and products. Given a reaction $\tau_1 = ([1,1,0],[0,0,2],k_1)$, we often refer to it as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$.

*CRN Semantics.* Under the usual assumption of mass action kinetics[1], the *stochastic* semantics of a CRN $\mathcal{N}$ is generally given in terms of a discrete-state, continuous-time stochastic process $\mathbf{X}(\mathbf{t}) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t), t \geq 0)$ [12]. The *state change* associated with the reaction $\tau$ is defined by $\upsilon_\tau = p_\tau - r_\tau$, i.e. the state $\mathbf{X}$ is changed to $\mathbf{X}' = \mathbf{X} + \upsilon_\tau$. For example, for $\tau_1$ as above, we have $\upsilon_{\tau_1} = [-1,-1,2]$. A reaction can only happen in a state $\mathbf{X}$ if all reactants are present in sufficient numbers. Then we say that the reaction is enabled in $\mathbf{X}$. The behavior of the stochastic system $\mathbf{X}(\mathbf{t})$ can be described by the (possibly infinite) continuous-time Markov chain (CTMC). The transition rate corresponding to a reaction $\tau$ is given by a *propensity function* that in general depends on the stoichiometry of reactants, their populations and the coefficient $k_\tau$.

### Related Concepts

*Population Level Abstraction.* The CTMC is the accurate representation of CRN $\mathcal{N}$, but—even when finite—it is not scalable in practice because of the state space explosion problem [20,25]. Various (adaptive) population abstractions [1,13,29,37] have been proposed to reduce the state-space and preserve the dynamics of the original CRN. Intuitively, *abstract states* are given by intervals on sizes of populations (with an additional specific that the abstraction captures enabledness of reactions). In other words, the population abstraction divides the *concrete states* of the CTMC into hyperrectangles called abstract states. We chose one concrete state within each abstract state as its *representative*. Although our approach is applicable to very general types of abstractions, for simplicity and specificity we consider in this paper only the *exponential partitioning* for some

[1] We can handle alternative kinetics including Michaelis-Menten and Hill kinetics.

parameter $1 < c \leq 2$ given as $\{[0,0]\} \cup \{[\lfloor c^{n-1} \rceil, \lfloor c^n \rceil - 1] : n \in \mathbb{N}\}$ for all dimensions. For example with $c = 2$ the intervals are $[0,0], [1,1], [2,3], [4,7], [8,15], \ldots$ i.e. they grow exponentially in $c$. While the structure of the abstract states is rather standard, the transitions between the abstract states are defined in different ways.

*Stochastic Simulation.* An alternative computational approach to the analysis of CRNs is to generate trajectories using stochastic algorithms for simulation. Gillespie's stochastic simulation algorithm (known as SSA) [15] is a widely used exact version of such algorithms, which produces statistically correct trajectories, i.e., sampled according to the stochastic process described by the Chemical Master Equation. To produce such a trajectory, SSA repeatedly applies one reaction at a time while keeping track of the elapsed time. This can take a long time if the number of reactions per trajectory is large. This is typically the case if (1) there are large numbers of molecules, (2) the system is stiff (with high differences in rates) or (3) we want to simulate the system for a time that is long compared to the rates of a single reaction. One of the approaches that mitigate the efficiency problem is $\tau$-leaping [16]. The main idea is that for a given time interval (of length $\tau$), where the reaction propensities do not change significantly, it is sufficient to sample (using Poisson distributions) only the number of occurrences for each reaction and not their concrete sequence. Having the numbers allows one to compute and apply the joint effect of the reactions at once. As detailed in the next section, instead of this time locality, we leverage a space locality.

## 3 The Plan: A Technical Overview

Since we shall work with different types of simulation runs, gradually building on top of each other, and both with the concrete system and its abstraction, we take the time here to overview the train of thoughts, the involved objects, and the four main conceptual steps:

– Section 4.1 introduces **segmental simulation** as a means to obtain simulation runs of the concrete system faster at the cost of (i) a significant memory overhead and (ii) skewing the probability space of the concrete runs, but only negligibly w.r.t. a user-given abstraction of the state space.
– Section 4.2 introduces **densely concrete simulation**, which eliminates the memory overhead but produces concrete simulation runs where only some of the concrete states on the run are known, however, frequently enough to get the full picture (again w.r.t. the abstraction), see Fig. 2 (bottom).
– Section 5.1 shows how to utilize Sect. 4 to equip the state-space abstraction (quotient) with a powerful transition function and semantics in terms of a probability space over abstract runs, i.e. runs over the abstract state space. The **abstraction is executable** and generates **abstract simulation runs** with extremely low memory requirements, yet allowing for transient analyses that are very precise (again w.r.t. the abstraction), see Fig. 3 (left).
– Section 5.2 considers the **concretization of abstract simulation runs** back to the concrete space, see Fig. 3 (right).

## 4 Segmental Simulation via Abstract States

### 4.1 Computing and Assembling Segments via Abstract States

**Precomputing Segments.** Assume we precomputed for each concrete state $s$ a list of $k$ randomly chosen short trajectories, called *segments*, starting in $s$. Note that we may precompute multiple segments with the same endpoint, reflecting that this evolution of the state is more probable. We can now obtain a trajectory of the system by repeatedly sampling and applying a precomputed segment for the current state instead of a single reaction.

**Using Abstraction.** While simulating with already precomputed segments would be faster, it is obviously inefficient to precompute and store the segments for each state separately. However, note that the rates of reactions in CRNs are similar for states with similar amounts of each species, in particular for states within the same abstract state of the population-level abstraction. Consequently, we only precompute $k$ segments for one concrete state per abstract state: the abstract state's representative (typically its center). For other states, the distribution of the segments would be similar and our approximation assumes them to be the very same. While the exponential population-level abstraction is a good starting point for many contexts, the user is free to provide any partitioning (quotient) of the state space that fits the situation and the desired granularity of the properties in question. E.g. one could increase the number of abstract states in regions of the state space we want to study.

An example for $k = 3$ precomputed segments is depicted in Fig. 1 (left). We choose to terminate each segment when it leaves the abstract state. Intuitively, at this point, at least one dimension changes significantly, possibly inducing significantly different rates.

**Assembling the Segments.** In a *segmental simulation*, instead of sampling a segment for the current concrete state, we sample a segment for the current representative. Because the sampled segment may start at a different state, we apply the relative effect of the segment to the current concrete state. Note that this is a conceptual difference compared with our previous work [9] where the segments are applied to abstract states. The importance of this difference is discussed in [21, Appendix C]. Figure 1 (right) illustrates the segmental simulation for the segments on the left. The system starts in an initial state, which belongs to the bottom left abstract state. Thus, segment $c$ was randomly chosen among the segments $a, b, c$ belonging to that abstract state. After applying the effect of segment $c$, the system is in the bottom right abstract state and thus we sample from $d, e, f$ and so on. Note that applying a segment might not change the current abstract state and might also only do so temporarily (like the application of $l$ and $h$, respectively, in the figure). Once the segment leaves the current state a different set of reactions might be enabled. Thus, to make sure that we never

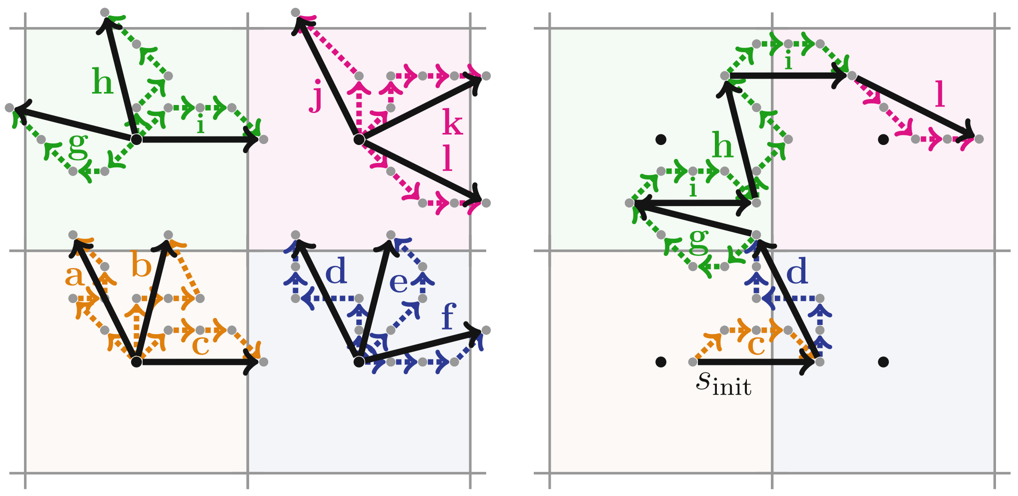


**Fig. 1.** (left) Four neighboring abstract states, drawn as squares. Each abstract state has $k = 3$ segments that start in their respective centers. Each segment is a sequence of reactions drawn as dotted arrows. The difference between the endpoint and the starting point is called a *summary* and is drawn in unbroken black. (right) A possible segmental simulation obtained by applying the segments $c, d, g, i, h, i$ and $l$ to the initial state $s_{\text{init}}$.

apply reactions that are not enabled, the population level abstraction has to satisfy some additional constraints.[2]

**Lazy and Adaptive Computation of Segments.** Instead of precomputing the segments for all abstract states, we generate them on the fly. When we need to sample the segments of an abstract state $a$ and there are less than $k$ segments, then we generate a new segment and store it. This new segment is the one we would have sampled from the $k$ (not yet computed) segments for $a$. Thus, we enlarge the reservoir of segments lazily, only as we need it. Since many abstract states might be rarely reached we generate only few segments for them if any at all. In contrast, we only generate many segments for frequently visited states, which are thus reused many times, improving the efficiency without much overhead. Note that segments can be reused already for a single simulation if that simulation visits the same abstract state more than $k$ times. However, the real benefit of our approach becomes apparent when we generate many simulations.

**Algorithm.** We summarize the approach in the pseudocode of Algorithm 1. It is already presented in a way that produces not one but $m$ simulations and computes the segments lazily. We start with no precomputed segments. As we simulate, we always compute the current abstract state $a$ (L. 6) and on L. 7 decide whether to simulate a new segment (L. 9) or uniformly choose from the previously computed ones (L. 11).

[2] We must choose a population abstraction such that applying any of the representative's possible segments to any corresponding concrete state may only change the enabledness of reactions with the last reaction. Similar constraints are needed if we want to avoid transitions to non-neighboring abstract states. For all presented models, the exponential population abstraction with $c \leq 2$ already has the desired properties.

**Algorithm 1:** Lazy Segmental Simulation

```
   Inputs : $\mathcal{N}$ (CRN), $k$ (number of segments), $c$ (partitioning parameter),
            $t_{\text{end}}$ (time horizon), $s_{\text{init}}$ (start state) and $m$ (number of simulations)
   Output: list of $m$ segmental simulations
1  $simulations := [\,]$;
2  $memory := \{\}$;                     // mapping each abstract state to a list of segments
3  for 1 to $m$ do
4      $s := s_{\text{init}}$;  $t := 0$;  $simulation := [(s,t)]$;
5      while $t < t_{end}$ do
6          a := abstractState$_c(s)$;
7          if $|memory(a)| < k$ then
8              $segment :=$ sampleNewSegment($a.representative$);       // sample new segment
9              $memory(a)$.add($segment$);                              // save it for reuse
10         else
11             $segment :=$ chooseUniformlyAtRandomFrom($memory(a)$);   // reuse old segment
12         end
           // apply segment's relative effects
13         $s := s + segment.\Delta_{\text{state}}$;  $t := t + segment.\Delta_{\text{time}}$;
14         $simulation$.add($(s,t)$);
15     end
16     $simulations$.add($simulation$);
17 end
18 return $simulations$
```

## 4.2 Densely Concrete Simulations

**Summaries.** Storing and applying the whole segments can still be memory- and time-intensive. Therefore, we replace each segment with a single "transition", called a *summary*. It captures the overall effect on the state, namely the difference between the end state and the starting state, and the time the sequence of reactions took. The summaries of segments in Fig. 1 (left) are depicted as solid black arrows. Algorithm 1 applies the segment's summary in L. 13.

**Assembling Summaries.** Instead of segments, we can append their summaries to the simulation runs, see Fig. 1 (right). We call the result a *densely concrete simulation*. The effect of this modification can be seen in Fig. 2. The top part of the figure shows a typical oscillating run of the predator-prey model that was produced via SSA simulation. The middle part displays a segmental simulation based on our abstraction that uses segments. It exhibits the expected oscillations with varying magnitudes and is visually indistinguishable from an SSA simulation. On the bottom, we see the corresponding densely concrete simulation where the segments of the same segmental simulation have been replaced with their summaries. We still observe the same global behavior but lose the local detail. More precisely, we only see those concrete states of the middle simulation that are the *seams of the segments* (now displayed as dots). All the other concrete states are unknown. However, they are close to the dots since they can only be in the same or neighboring abstract states, meaning the known concrete states are arranged *densely* enough. Moreover, the distance between the dots corresponds to the lengths of the segments. Hence the dots are arranged sparsely only if the system entered a very stable abstract state. Altogether, all changes are reflected faithfully, relative to the level of detail of the abstraction.

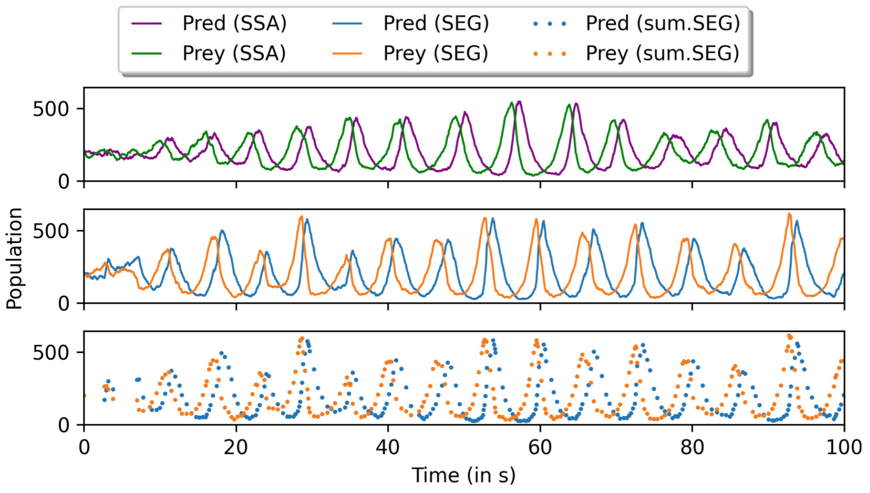


**Fig. 2.** Comparison of simulations for the predator-prey system: SSA simulation (top), segmental simulation (middle) and the same segmental simulation as densely concrete simulation where segments are replaced with their summaries (bottom).

### 4.3 Introduced Inaccuracy

We summarize the sources of errors our approach introduces.

**(1) Number of Segments.** Instead of sampling from all of the possible trajectories, we only sample from $k$ segments and consequently lose some variance. Thus, if $k$ is too small or if the trajectories are too long, our abstraction might miss important behavior of the original system. However, it is easy to see that this error vanishes for $k \to \infty$. It is thus crucial to choose an appropriate value for $k$ and we discuss this choice in Sect. 6. However, note that sampling from segments instead of reactions cannot produce spurious behavior. In other words, all trajectories we obtain by sampling segments are possible trajectories of the original system. Further, if the segments we sample from are representative enough of the actual distribution of trajectories, we will exhibit the same global behavior.

**(2) Size of the Abstract States.** Recall that we do not sample the distribution of segments for the current state but instead sample the distribution for the representative. Because the propensities and thus the rates of the reactions are different in the current state and the representative state, this inherently introduces an error. However, this error is small if we assume that the distribution over segments does not significantly change within the abstract state. This assumption is reasonable since the propensity functions and thus the rates of the reactions are similar for similar populations and change only slowly; except when the number of molecules is close to zero but there the exponential abstraction provides very fine abstract states. Further, we can decrease the parameter $c$ that determines the interval sizes of the exponential population-level abstraction. A discussion of the influence of parameter $c$ on the accuracy of our method can be found in Sect. 6.

## 5 From Segmental Simulations to Abstract Simulations

In this section, we focus on abstract simulation runs, i.e. runs over the abstract state space, for several reasons. Concrete simulation runs, i.e. runs over the concrete state space, provide a rich piece of information about the system, however, already storing a large number of very long simulation runs may be infeasible. Compared to segmental simulation, densely concrete simulation drops most concrete states by only remembering the seams of the segments; yet the number of concrete states can be large and each one can take non-trivial space if the populations reach large numbers. Another disadvantage is that, for most of the time points, we only know that the current state is in the same abstract state as the nearest seams or in their neighbors, but no exact concrete state. In contrast, the abstraction may hide non-interesting details and instead show the big picture. Altogether, if only population levels are of interest, abstract simulations can be useful.

### 5.1 Segmental Abstraction of CTMC and Abstract Simulation

Population-level abstraction of the CTMC might be a lot more explainable than the complete CTMC. However, while the state space of the population abstraction of a CTMC is simply given by the population levels as the Cartesian product of the intervals over all the species, it is not clear what the nature of the transitions should be. There are two issues the previous literature faces. First, what should be the dynamics of an abstract state if each concrete state within behaves a bit differently? Second, what should be the dynamics of one abstract step between two abstract states when it corresponds to a varying number of concrete steps? Standard approaches either pick a representative state and copy its dynamics, e.g. the rate of the reaction, or take an over-approximation of all behaviors of all possible members of the class, e.g. take an interval of possible rates.

Here we reuse the segmental simulation and the concepts of Sect. 4 to formally define a transition function on the abstraction. This gives us the abstraction's semantics and makes it executable. Moreover, the resulting behavior is close to the original system (in contrast to, e.g. [9], we can even preserve the oscillations of the predator-prey models), but at the expense of making the abstract model non-Markovian. Intuitively, our *segmental abstraction* of the CTMC is given by the abstract state space and the segments, exactly as depicted in Fig. 1 (left). Similar to other non-Markovian systems, the further evolution of the system is not given only by its current state, but also by some information about the history of the run so far. For instance, in the case of semi-Markov processes, it is the time spent in the state so far; in the case of generalized semi-Markov processes, it is the times each event has been already scheduled for. In the case of the segmental abstraction, it is a vector forming a concrete state.

Formally, the *configuration* of the segmental abstraction is a triple $(a, s, t)$ where $a$ is an abstract state, $s$ one of its concrete states, and $t$ a time. The probability to move from $(a, s, t)$ to $(a', s', t')$ is then given by the probability to sample a segment with a summary $s' - s$ on states and taking $t' - t$ time. (Hence we can store the summaries only, as described in Sect. 4.2.) Given a concrete

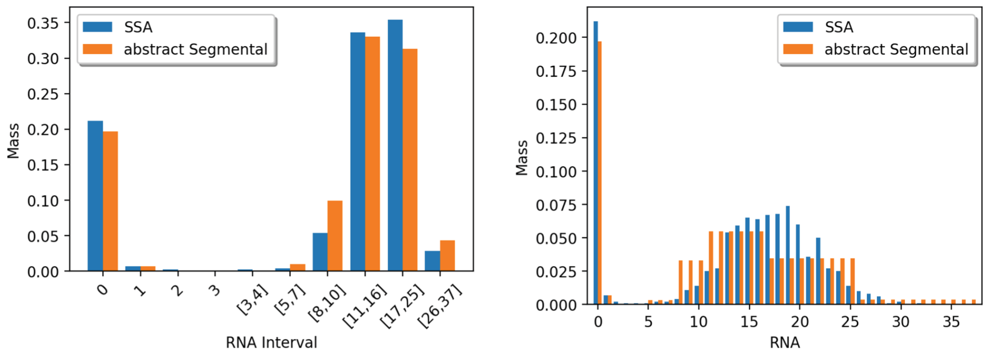


**Fig. 3.** RNA distribution in the viral infection model at $t = 200\,\text{s}$ predicted by SSA and abstract segmental simulation with $c = 1.5$ and $k = 100$: in the abstract domain (left) and the concrete domain (right) where the segmental simulation's abstract values were concertized using a uniform distribution over the interval.

state $s$ to start in, there is a unique probability space over the abstract runs initiated in $(a, s, 0)$ obtained by dropping the second component.

The probability space coincides with the probability space introduced by the segmental simulation (in the variant with summaries of precomputed segments) when the concrete runs are projected by the population-level abstraction to the abstract runs. However, (i) the space needed to store the abstract simulations is smaller and (ii) transient analysis is well defined for every time point, while its results are still very faithful. Indeed, Fig. 3 (left) shows an example of the transient analysis (at a given time point $t$) obtained by (i) the states reached by real simulation runs and clustered according to the population-level abstraction, and (ii) abstract states reached by abstract segmental simulation runs. Given the granularity of the abstraction, the results are very close.

### 5.2 From Abstract Simulations Back to Concrete Predictions

Further, one can map the abstract states to sets of concrete states. Consequently, the results of the abstract transient analysis can be mapped to a distribution over concrete states, whenever we assume a distribution over the concrete states corresponding to one abstract state. For instance, taking uniform distribution as a baseline, we obtain a concrete transient analysis from the abstract one, see Fig. 3 (right), which already shows a close resemblance.

## 6 Experimental Evaluation

We evaluate the densely concrete version of segmental simulation and consider the following three research questions:

Q1 What is the accuracy of segmental simulation?
Q2 What are the trade-offs between accuracy and performance?
Q3 Are the achieved trade-offs competitive with alternative approaches?

### Experimental Setting and Accuracy Measurement

*Benchmark selection.* We use the following models from the literature: (1) Viral Infection (VI) [35], (2) Repressilator (RP) [24], (3) Toggle Switch (TS) [24] and (4) Predator-Prey (PP, a.k.a. Lotka-Volterra) [15]. The formal definition for each of these models can be found in [21, Appendix D]. Although the underlying CRNs are quite small (up to 6 species and 15 reactions), their analysis is very challenging due to stochasticity, multi-scale species populations and stiffness. Therefore, the models are commonly used to evaluate advanced numerical as well as simulation methods.

*Implementation and HW Configuration.* Our approach is implemented as modification of SeQuaiA, a Java-based tool for semi-quantitative analysis of CRNs [8]. All experiments run on a 1.80 GHz Lenovo ThinkPad T580 with 8 GB of RAM. To report speedups, we use our own competitive implementation of the SSA method as a baseline. Depending on the model, it achieves between $1{\times}10^6$ and $7{\times}10^6$ reactions per second. We refer to the SeQuaiA repository[3] for all active development and provide an artefact[4] to reproduce the experimental results.

*Assessing Accuracy.* To measure the accuracy, we compare transient distributions for one species at a time. For this, we approximate the implied transient distribution of our approach and of SSA by running a large number of simulations. We used 1000 simulations for the models RP and TS as their simulations take longer, and 10000 for PP and VI. The resulting histograms for the studied species are then normalized to approximate the transient distribution of that species. To quantify the error, we compare the means of both distributions and report the earth-mover-distance (EMD) between them. Because EMD values are difficult to interpret without context, we additionally report the EMD between two different transient distributions that were computed with SSA. Intuitively, even if segmental simulation was as accurate as SSA, we would expect to see a EMD similar to the EMD of this "control SSA". Additionally, Fig. 12 in [21, Appendix B] compares the variance.

### Q1 What is the Accuracy of Segmental Simulation?

In this section, we evaluate the accuracy of the segmental simulation scheme and the effects of parameters $c$ (size of the abstraction) and $k$ (the number of stored summaries) in a quantitative manner. For a more qualitative evaluation see Fig. 2 and [21, Appendix A] where you find exemplary simulations and trajectories for both segmental simulation and SSA. We consider three population abstractions given by $c \in \{2, 1.5, 1.3\}$ (recall that $c = 2$ is the most coarse abstraction as explained in Sect. 2) and three values of $k$, namely $k \in \{10, 100, 1000\}$.

We start with the VI model where one is typically interested in the distribution of the RNA population at a given time $t$. Figure 4 (left) shows the

[3] https://sequaia.model.in.tum.de (SeQuaiA).
[4] https://doi.org/10.5281/zenodo.6658924 (artifact).

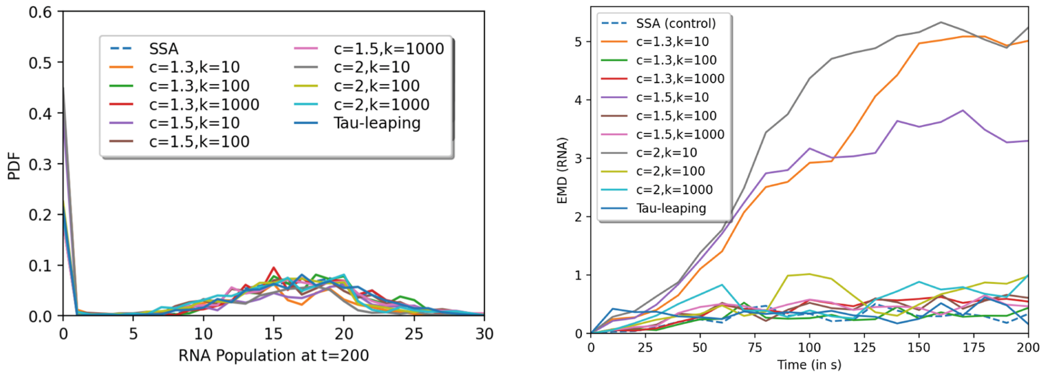


**Fig. 4.** Accuracy on the viral infection model using different abstractions.

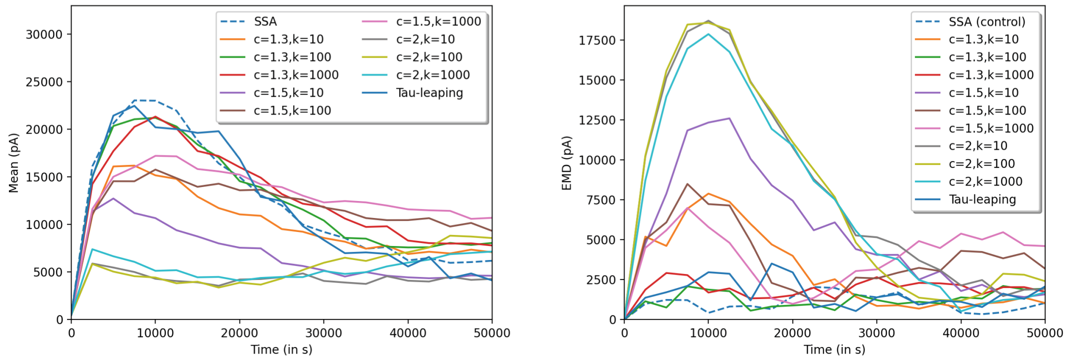
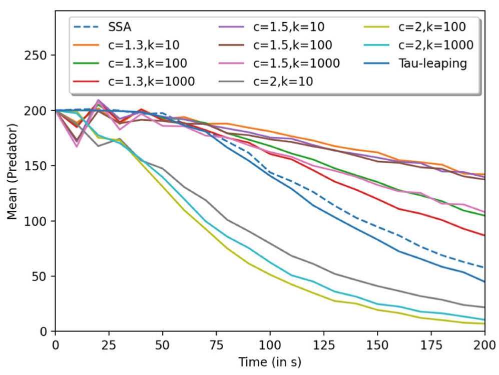


**Fig. 5.** Accuracy on the repressilator model using different abstractions.

distributions at $t = 200$ obtained by SSA simulation and by segmental simulations with different values of the parameters $c$ and $k$. We observe that all distributions show the expected bi-modality [17]. If a less precise abstraction is used ($c = 2$ and/or $k = 10$), the probability that RNA dies out is significantly higher than the reference value. In Fig. 4 (right), we evaluate how the EMD (for the RNA) changes in time for particular settings. The results clearly confirm that $k = 10$ leads to significant inaccuracy. For all other settings, the EMD is very close to the SSA control demonstrating the very high accuracy of our approach. The only notable exceptions are the variants with $c = 2$, where the EMD fluctuates. We also observe that increasing $k$ from 100 to 1000 does not bring any considerable improvement.

Different trends can be observed for the RP model. Figure 5 shows how the mean value (left) and the EMD (right) of the species pA change over time. We observe that the partitioning of the populations plays a more important role here, i.e., the coarse abstraction ($c = 2$) induces a notable inaccuracy. The fact that the accuracy is less sensitive with respect to the low values of $k$ is a result of the very regular dynamics of the model where the populations of the proteins pA and pB oscillate and slowly decrease. Similar trends (not presented here) are observed also for the TS model.

Finally, we consider the PP model. Although very simple, it is notoriously difficult for abstraction-based approaches since they struggle to preserve the



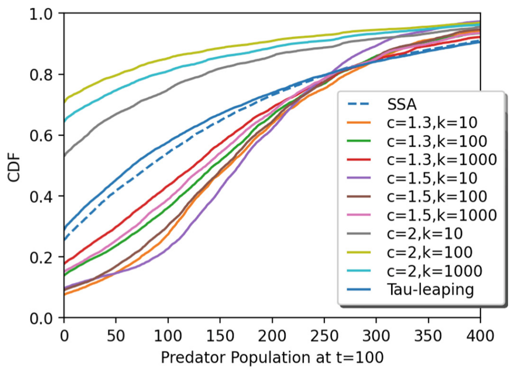
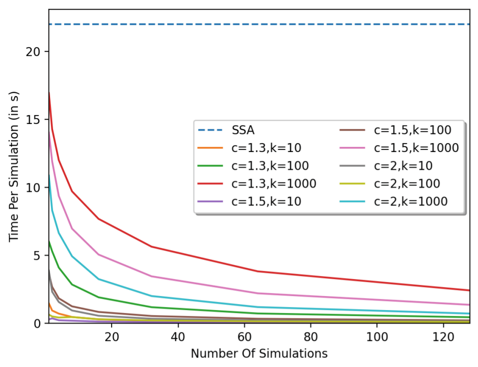


**Fig. 6.** Accuracy on the predator-prey model using different abstractions.



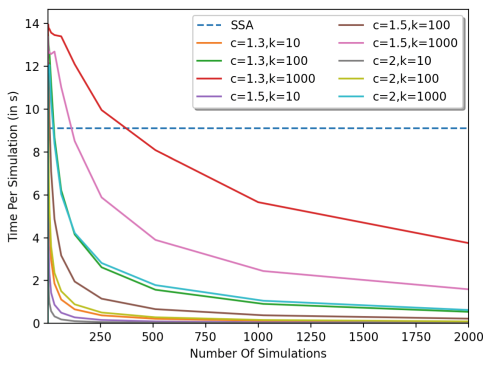


**Fig. 7.** Increasing speed of lazy segmental simulation for the toggle switch model (left) and repressilator model (right).

oscillation and the die-out time. Recall Fig. 2 of Sect. 4 where we clearly observe the expected oscillation with the correct frequency in a segmental simulation for $c = 2$ and $k = 100$. Figure 6 (left) shows how the mean population of the predators changes in time. We observe that the less precise abstractions do not accurately preserve the rate at which the mean population decreases. On the other hand, the most precise setting is close to the SSA reference and control curve. Figure 6 (right) shows the cumulative predator distributions at $t = 100$ demonstrating how the simulations using less precise abstractions deviate from the reference solution.

## Q2 What are the Trade-Offs Between Accuracy and Performance?

Recall that our approach is based on re-using the segments generated in the previous simulation runs. Figure 7 shows how the average time per simulation decreases for a growing number of simulations. For some models (like TS) segmental simulation is always faster than SSA because it can reuse segments already during the first simulation. For other models (like RP), segmental simulation becomes faster after a number of simulations that depends on the precision of the abstraction. Table 1 shows the speedup factor we observe when running 10,000 segmental simulations instead of SSA simulations. Table 5 in [21, Appendix B] reports the average

**Table 1.** Average run-time of one SSA simulation and the speedup factor of segmental simulation when computing 10,000 simulations with different abstraction parameters.

| Model | SSA | SEG $k = 10$ | | | SEG $k = 100$ | | | SEG $k = 1000$ | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ |
| PP | 0.014 s | 70x | 70x | 70x | 70x | 70x | 23x | 28x | 23x | 12x |
| VI | 0.88 s | 730x | 380x | 180x | 100x | 48x | 17x | 8.6x | 4.8x | 2.9x |
| TS | 22 s | 360x | 360x | 340x | 390x | 350x | 280x | 250x | 190x | 110x |
| RP | 9.1 s | 760x | 540x | 320x | 300x | 140x | 62x | 54x | 21x | 7.4x |

**Table 2.** Number of visited abstract states after 10,000 segmental simulations for different abstraction parameters.

| Model | SEG $k = 10$ | | | SEG $k = 100$ | | | SEG $k = 1000$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ |
| PP | 163 | 398 | 832 | 170 | 391 | 888 | 170 | 397 | 885 |
| VI | 1,022 | 3,669 | 1,0337 | 1,269 | 3,797 | 11,524 | 1,353 | 4,018 | 11,218 |
| TS | 5,072 | 13,167 | 38,827 | 9,248 | 25,733 | 65,259 | 10,388 | 29,424 | 74,950 |
| RP | 12,921 | 42,241 | 124,280 | 18,014 | 56,312 | 155,394 | 21,460 | 64,385 | 146,126 |

number of reactions per SSA simulation and the average number of applied summaries per segmental simulation. Comparing these numbers highlights the source of the speedup: instead of sampling and applying many reactions, segmental simulation does the same for fewer summaries. This also gives an estimate for the speedup factor that can be reached once only precomputed summaries are used. Note that the speedup factors we report are smaller because we include the time needed for computing the summaries.

The PP model includes only two species and it is not stiff. Therefore, already the SSA simulation is quite fast. Our approach still achieves a stable speedup around 70x that drops to a factor of 12x for the most precise abstraction ($c = 1.3$ and $k = 1000$) providing accuracy close to the control SSA.

For the VI model, we observe a slowdown of the segmental simulation when improving the abstraction, namely, for $k = 1000$. Recall, we reported very good accuracy already for $c = 2$ and $k = 100$ which gives us a speedup factor of 100x. For an accuracy that is close to the control SSA, we archive a speedup factor of 50x.

The TS model exhibits regular oscillation, where a typical run repeatedly visits the same abstract states. This is very beneficial for our approach as we can very efficiently reuse segments. We observe a significant slowdown for $c = 1.3$ and $k = 1000$, since reusing segments is much less effective. The RP model has similar characteristics, but we observe an even more significant slowdown when the abstraction is refined. As we discussed in the previous section, very good accuracy for these models is achieved already for $c = 1.5$ and $k = 100$ which give us speedup factors 350x and 140x for the TS and RP model, respectively.

In general, segmental simulation is never slower than SSA by more than the constant factor that is the result of saving and loading segments and it eventually becomes faster if we compute enough simulations. We pay for the inevitable

**Table 3.** Size of segmental abstraction after 10,000 simulations for different parameters.

| Model | SEG $k = 10$ | | | SEG $k = 100$ | | | SEG $k = 1000$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ | $c = 2$ | $c = 1.5$ | $c = 1.3$ |
| PP | 25 KB | 61 KB | 130 KB | 250 KB | 570 KB | 1.3 MB | 2.2 MB | 4.8 MB | 11 MB |
| VI | 210 KB | 730 KB | 2.0 MB | 1.8 MB | 4.8 MB | 13 MB | 11 MB | 25 MB | 53 MB |
| TS | 1.2 MB | 3.0 MB | 8.7 MB | 15 MB | 37 MB | 85 MB | 100 MB | 250 MB | 550 MB |
| RP | 3.8 MB | 12 MB | 34 MB | 43 MB | 120 MB | 300 MB | 310 MB | 760 MB | 1.0 GB |

speedup with increased memory consumption. Table 2 shows how the number of visited abstract states after 10,000 segmental simulations. The number of saved summaries [21, Appendix B, Table 6] is approximately the number of visited abstract states times $k$. The memory consumed by one summary is an integer vector and a floating-point number that describe the effect on state and time, respectively. The size of the abstraction is shown in Table 3.

We observe a trade-off between accuracy and performance: smaller abstract states results in segments with fewer steps and thus slower simulations and for more precise abstractions we need to save more summaries. Further, we can only reuse simulations if they visit the same abstract states. This implies that the presented approach does not scale favorably with the dimension of the studied system. To handle many species, one can use the available memory only for the most important abstract states, e.g. the most visited ones, and simulate normally in all other regions. But there is an inherent trade-off between memory consumption and simulation speed as we can only reuse segments if we save them.

## Q3: Comparison with Alternative Approaches

*Comparison with $\tau$-Leaping.* We first compare the performance and accuracy of our approach with the $\tau$-leaping method implemented in StochPy [28], a widely used stochastic modeling and simulation package. $\tau$-leaping achieves very good accuracy on the considered models. Quantitatively, it is very close to the SSA control runs and typically provides slightly better results than our best setting (compare Figs. 4, 5 and 6). On the other hand, we typically observe only a moderate speedup (around one order of magnitude) with respect to the SSA. Note that a direct comparison with our run-times is unfair as it is known that StochPy uses an inefficient python-based random number generator that can significantly slow down the simulation. For example, a single SSA simulation of the RP model takes in StochPy 1000 s and $\tau$-leaping achieves a 16-times speedup while our SSA baseline takes around 9 s and the segmental simulation achieves the speedup of a factor over 140 (to our baseline) with only a small drop in the accuracy. On the other hand, for the PP model, $\tau$-leaping provides only a negligible speedup (below factor 2). Recall we observed a speedup factor between 12x and 70x depending on the required precision.

*Comparison with Advanced Simulation Methods.* A fair comparison with slow-scale stochastic simulations [4, 17, 32] is problematic since, to our best knowledge,

**Table 4.** Runtime comparison with [24] for 100,000 simulations.

| Model | results presented in [24] | | | Our results | | |
|---|---|---|---|---|---|---|
| | SSA | Adaptive hybrid | Speedup | SSA | SEG($c = 1.5,\ k = 100$) | Speedup |
| RP | 232 hours | 3 hours | 77x | 252 hours | 1.8 hours | 140x |
| TS | 47 days | 1.1 days | 43x | 25 days | 0.07 days | 350x |

there is no available implementation. Therefore, we focus on the comparison with the results presented in [24] representing the state-of-the-art hybrid simulation method. In Table 4, we compare the run-times of the adaptive hybrid simulation (100,000 runs) on the repressilator and toggle switch models reported in [24] (Fig. 1) with our approach. We report the runtimes only for the setting with $c = 1.5$ and $k = 100$ which already leads to very good accuracy for these models. The table also shows runtimes for the baseline SSA and the achieved speedup factor to make the comparison between the different hardware configurations fair. We observe that a significant computational gain (over two orders of magnitude) is achieved by our approach.

*Comparison with the Deep Learning Approaches.* Finally, we compare with the approach of [3] where a neural network is trained to provide a fast and accurate generator of simulations in the original CRN. To this end, we use a simpler variant of the toggle switch model considered in [3]. If we run more than 1000 simulations, a single simulation takes on average 0.0004 s which is comparable with the neural-based generator (the authors report 0.0008 s per simulation). Regarding the accuracy, we achieve comparable values of the EMD (note the EMD is scaled in [3]). The key benefit of our approach, however, lies in the fact that it does not require the computationally very demanding training phase.

## 7 Conclusion and Future Directions

We have proposed a novel simulation scheme enabling us to efficiently generate a large number of simulation runs for complex CRNs. It is based on reusing segments of the runs computed over abstract states but applied to concrete states. Already our initial experiments demonstrate that the simulation scheme preserves key dynamical and quantitative properties while providing a significant computational gain over the existing approaches. On the conceptual level, we define an executable abstraction of the CTMC, preserving the dynamics very faithfully. In particular, we have the machinery to generate abstract simulation runs, which take less space than the concrete ones, yet provide high precision on the level of detail given by the population levels defined by the user.

In future work, we want to investigate the error with the goal of giving formal error bounds. Further, we propose an adaptive version of the abstraction where the population abstraction and the number of precomputed segments are refined or learned. Alternatively, instead of memorizing a discrete distribution over precomputed segments, we can generalize to unobserved behavior by learning some continuous distribution.

Segmental simulation via abstract states can be understood as a general framework for accelerating the simulation of population models. As such, it can be combined with any method that predicts the evolution of such models. In particular, it can naturally leverage an adaptive multi-scale approach where different simulation techniques are used in different regions of the state-space.

## References

1. Abate, A., Andriushchenko, R., Češka, M., Kwiatkowska, M.: Adaptive formal approximations of Markov chains. Perform. Eval. **148**, 102207 (2021)
2. Backenköhler, M., Bortolussi, L., Großmann, G., Wolf, V.: Abstraction-guided truncations for stationary distributions of Markov population models. In: Abate, A., Marin, A. (eds.) QEST 2021. LNCS, vol. 12846, pp. 351–371. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85172-9_19
3. Cairoli, F., Carbone, G., Bortolussi, L.: Abstraction of Markov population dynamics via generative adversarial nets. In: Cinquemani, E., Paulevé, L. (eds.) CMSB 2021. LNCS, vol. 12881, pp. 19–35. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85633-5_2
4. Cao, Y., Gillespie, D.T., Petzold, L.R.: The slow-scale stochastic simulation algorithm. J. Chem. Phys. **122**(1), 014116 (2005)
5. Cao, Y., Gillespie, D.T., Petzold, L.R.: Efficient step size selection for the tau-leaping simulation method. J. Chem. Phys. **124**(4), 044109 (2006)
6. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**(02), 247–271 (2013)
7. Cardelli, L., Kwiatkowska, M., Laurenti, L.: A stochastic hybrid approximation for chemical kinetics based on the linear noise approximation. In: Bartocci, E., Lio, P., Paoletti, N. (eds.) CMSB 2016. LNCS, vol. 9859, pp. 147–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45177-0_10
8. Češka, M., Chau, C., Křetínský, J.: SeQuaiA: a scalable tool for semi-quantitative analysis of chemical reaction networks. In: Lahiri, S.K., Wang, C. (eds.) CAV 2020. LNCS, vol. 12224, pp. 653–666. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-53288-8_32
9. Češka, M., Křetínský, J.: Semi-quantitative abstraction and analysis of chemical reaction networks. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 475–496. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_28
10. Chellaboina, V., Bhat, S.P., Haddad, W.M., Bernstein, D.S.: Modeling and analysis of mass-action kinetics. IEEE Control Syst. Mag. **29**(4), 60–78 (2009)
11. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: logic, simulation and games. In: 2008 Fifth International Conference on Quantitative Evaluation of Systems, pp. 264–273. IEEE (2008)
12. Ethier, S.N., Kurtz, T.G.: Markov Processes: Characterization and Convergence, vol. 282. Wiley, Hoboken (2009)
13. Ferm, L., Lötstedt, P.: Adaptive solution of the master equation in low dimensions. Appl. Numer. Math. **59**(1), 187–204 (2009)

14. Ganguly, A., Altintan, D., Koeppl, H.: Jump-diffusion approximation of stochastic reaction dynamics: error bounds and algorithms. Multisc. Model. Simul. **13**(4), 1390–1419 (2015)
15. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
16. Gillespie, D.T.: Approximate accelerated stochastic simulation of chemically reacting systems. J. Chem. Phys. **115**(4), 1716–1733 (2001)
17. Goutsias, J.: Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. J. Chem. Phys. **122**(18), 184102 (2005)
18. Gupta, A., Schwab, C., Khammash, M.: DeepCME: a deep learning framework for computing solution statistics of the chemical master equation. PLoS Comput. Biol. **17**(12), e1009623 (2021)
19. Hasenauer, J., Wolf, V., Kazeroonian, A., Theis, F.: Method of conditional moments (MCM) for the chemical master equation. J. Math. Biol. **69**, 1–49 (2013)
20. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoret. Comput. Sci. **391**(3), 239–257 (2008)
21. Helfrich, M., Češka, M., Křetínský, J., Martiček, Š.: Abstraction-based segmental simulation of chemical reaction networks. arXiv (2022). https://doi.org/10.48550/arXiv.2206.06677
22. Henzinger, T.A., Mateescu, M., Wolf, V.: Sliding window abstraction for infinite Markov chains. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 337–352. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_27
23. Henzinger, T.A., Mikeev, L., Mateescu, M., Wolf, V.: Hybrid numerical solution of the chemical master equation. In: CMSB 2010, pp. 55–65. ACM (2010)
24. Hepp, B., Gupta, A., Khammash, M.: Adaptive hybrid simulations for multiscale stochastic reaction networks. J. Chem. Phys. **142**(3), 034118 (2015)
25. Kwiatkowska, M., Thachuk, C.: Probabilistic model checking for biology. Softw. Syst. Saf. **36**, 165 (2014)
26. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
27. Lester, C., Yates, C.A., Giles, M.B., Baker, R.E.: An adaptive multi-level simulation algorithm for stochastic biological systems. J. Chem. Phys. **142**(2), 01B612_1 (2015)
28. Maarleveld, T.R., Olivier, B.G., Bruggeman, F.J.: StochPy: a comprehensive, user-friendly tool for simulating stochastic biological processes. PLoS One **8**(11), e79345 (2013)
29. Madsen, C., Myers, C., Roehner, N., Winstead, C., Zhang, Z.: Utilizing stochastic model checking to analyze genetic circuits. In: Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 379–386. IEEE (2012)
30. Mateescu, M., Wolf, V., Didier, F., Henzinger, T.A.: Fast adaptive uniformization of the chemical master equation. IET Syst. Biol. **4**(6), 441–452 (2010)
31. Munsky, B., Khammash, M.: The finite state projection algorithm for the solution of the chemical master equation. J. Chem. Phys. **124**, 044104 (2006)
32. Rao, C.V., Arkin, A.P.: Stochastic chemical kinetics and the quasi-steady-state assumption: application to the Gillespie algorithm. J. Chem. Phys. **118**(11), 4999–5010 (2003)
33. Salis, H., Kaznessis, Y.: Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. J. Chem. Phys. **122**(5), 054103 (2005)

34. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. U.S.A. **107**(12), 5393–5398 (2010)
35. Srivastava, R., You, L., Summers, J., Yin, J.: Stochastic vs. deterministic modeling of intracellular viral kinetics. J. Theor. Biol. **218**(3), 309–321 (2002)
36. Van Kampen, N.G.: Stochastic Processes in Physics and Chemistry, vol. 1. Elsevier, Amsterdam (1992)
37. Zhang, J., Watson, L.T., Cao, Y.: Adaptive aggregation method for the chemical master equation. Int. J. Comput. Biol. Drug Des. **2**(2), 134–148 (2009)

# Part II

# Non-first Author Publications

# D Succinct Population Protocols for Presburger Arithmetic

This chapter has been published as a **peer-reviewed conference paper**.

Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. "Succinct Population Protocols for Presburger Arithmetic". In: *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Ed. by Christophe Paul and Markus Bläser. Vol. 154. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 40:1–40:15. DOI: 10.4230/LIPIcs.STACS.2020.40

**Summary.** We present the first synthesis procedure that results in succinct population protocols, i.e., protocols with agents that have few states. Specifically, for a given Presburger formula $\varphi$, we produce a population protocol with agents that have poly($|\varphi|$) states where $|\varphi|$ is the length of the formula with binary coefficients. Our construction has two parts: The first part handles large inputs. It produces succinct protocols with $\mathcal{O}(|\varphi|^3)$ leaders, special agents that are independent of the input. Because the input is large, these leaders can be simulated by normal agents resulting in a leaderless population protocol. The second part handles small inputs, which makes it possible to design protocols that compute in a sequence of steps.

**Contributions of thesis author.** The author made valuable contributions to the manuscript and played a significant role in the development of the results presented in the paper. In addition, they actively participated in discussions and provided feedback during the revision process. Noteworthy individual contributions are the initial idea that initiated the endeavor, the design of the protocols for large inputs in the first half of the paper, as well as their visualization in Figure 1.

# Succinct Population Protocols for Presburger Arithmetic

**Michael Blondin**
Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca

**Javier Esparza**
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
esparza@in.tum.de

**Blaise Genest**
Univ Rennes, CNRS, IRISA, France
blaise.genest@irisa.fr

**Martin Helfrich**
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
helfrich@in.tum.de

**Stefan Jaax**
Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
jaax@in.tum.de

## Abstract

In [5], Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA), the first-order theory of addition. As part of this result, they presented a procedure that translates any formula $\varphi$ of quantifier-free PA with remainder predicates (which has the same expressive power as full PA) into a population protocol with $2^{\mathcal{O}(\text{poly}(|\varphi|))}$ states that computes $\varphi$. More precisely, the number of states of the protocol is exponential in both the bit length of the largest coefficient in the formula, and the number of nodes of its syntax tree.

In this paper, we prove that every formula $\varphi$ of quantifier-free PA with remainder predicates is computable by a leaderless population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Our proof is based on several new constructions, which may be of independent interest. Given a formula $\varphi$ of quantifier-free PA with remainder predicates, a first construction produces a succinct protocol (with $\mathcal{O}(|\varphi|^3)$ leaders) that computes $\varphi$; this completes the work initiated in [8], where we constructed such protocols for a fragment of PA. For large enough inputs, we can get rid of these leaders. If the input is not large enough, then it is small, and we design another construction producing a succinct protocol with *one* leader that computes $\varphi$. Our last construction gets rid of this leader for small inputs.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models; Theory of computation → Automata over infinite objects; Theory of computation → Logic and verification

**Keywords and phrases** Population protocols, Presburger arithmetic, state complexity

**Digital Object Identifier** 10.4230/LIPIcs.STACS.2020.40

**Related Version** A full version of this paper is available at [7], `https://arxiv.org/abs/1910.04600`.

**Funding** *Javier Esparza, Martin Helfrich, Stefan Jaax*: Supported by an ERC Advanced Grant (787367: PaVeS).
*Michael Blondin*: Supported by a Quebec–Bavaria project funded by the Fonds de recherche du Québec (FRQ), by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Fonds de recherche du Québec – Nature et technologies (FRQNT).

## 1 Introduction

Population protocols [3, 4] are a model of distributed computation by indistinguishable, mobile finite-state agents, intensely investigated in recent years (see e.g. [2, 10]). Initially introduced to model networks of passively mobile sensors, they have also been applied to the analysis of chemical reactions under the name of chemical reaction networks (see e.g. [14]).

In a population protocol, a collection of agents, called a *population*, randomly interact in pairs to decide whether their initial configuration satisfies a given property, e.g. whether there are initially more agents in some state $A$ than in some state $B$. Since agents are indistinguishable and finite-state, their configuration at any time moment is completely characterized by the mapping that assigns to each state the number of agents that currently populate it. A protocol is said to *compute a predicate* if for every initial configuration where the predicate holds, the agents eventually reach consensus 1, and they eventually reach consensus 0 otherwise.

In a seminal paper, Angluin et al. proved that population protocols compute exactly the predicates definable in Presburger arithmetic (PA) [5]. As part of the result, for every Presburger predicate Angluin et al. construct a leaderless protocol that computes it. The construction uses the quantifier elimination procedure for PA: every Presburger formula $\varphi$ can be transformed into an equivalent boolean combination of *threshold predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} > \beta$ and *remainder predicates* of the form $\boldsymbol{\alpha} \cdot \boldsymbol{x} \equiv \beta \pmod{m}$, where $\boldsymbol{\alpha}$ is an integer vector, and $\beta, m$ are integers [12]. Slightly abusing language, we call the set of these boolean combinations *quantifier-free Presburger arithmetic* (QFPA)[1]. Using that PA and QFPA have the same expressive power, Angluin et al. first construct protocols for all threshold and remainder predicates, and then show that the predicates computed by protocols are closed under negation and conjunction.

The construction of [5] is simple and elegant, but it produces large protocols. Given a formula $\varphi$ of QFPA, let $n$ be the number of bits of the largest coefficient of $\varphi$ in absolute value, and let $m$ be the number of atomic formulas of $\varphi$, respectively. The number of states of the protocols of [5] grows exponentially in both $n$ and $m$. In terms of $|\varphi|$ (defined as the sum of the number of variables, $n$, and $m$) they have $\mathcal{O}(2^{\text{poly}(|\varphi|)})$ states. This raises the question of whether *succinct protocols* with $\mathcal{O}(\text{poly}(|\varphi|))$ states exist for every formula $\varphi$ of QFPA. We give an affirmative answer by proving that every formula of QFPA has a succinct and leaderless protocol.

Succinct protocols are the state-complexity counterpart of *fast protocols*, defined as protocols running in polylogarithmic parallel time in the size of the population. Angluin et al. showed that every predicate has a fast protocol with a leader [6], but Alistarh et al., based on work by Doty and Soloveichik [9], proved that in the leaderless case some predicates need linear parallel time [1]. Our result shows that, unlike for time complexity, succinct protocols can be obtained for every QFPA formula in both the leaderless case and the case with leaders.

The proof of our result overcomes a number of obstacles. Designing succinct leaderless protocols is particularly hard for inputs with very few input agents, because there are less resources to simulate leaders. So we produce two completely different families of protocols, one for small inputs with $\mathcal{O}(|\varphi|^3)$ agents, and a second for large inputs with $\Omega(|\varphi|^3)$ agents, and combine them appropriately.

[1] Remainder predicates cannot be directly expressed in Presburger arithmetic without quantifiers.

**Large inputs.** The family for large inputs is based on our previous work [8]. However, in order to obtain leaderless protocols we need a new succinct construction for boolean combinations of atomic predicates. This obstacle is overcome by designing new protocols for threshold and remainder predicates that work under *reversible dynamic initialization*. Intuitively, agents are allowed to dynamically "enter" and "leave" the protocol through the initial states (dynamic initialization). Further, every interaction can be undone (reversibility), until a certain condition is met, after which the protocol converges to the correct output for the current input. We expect protocols with reversible dynamic initialization to prove useful in other contexts, since they allow a protocol designer to cope with "wrong" non-deterministic choices.

**Small inputs.** The family of protocols for small inputs is designed from scratch. We exploit that there are few inputs of small size. So it becomes possible to design one protocol for each possible size of the population, and combine them appropriately. Once the population size is fixed, it is possible to design agents that check if they have interacted with all other agents. This is used to simulate the *concatenation operator* of sequential programs, which allows for boolean combinations and succinct evaluation of linear combinations.

**Relation to previous work.** In [8], we designed succinct protocols with leaders for systems of linear equations. More precisely, we constructed a protocol with $\mathcal{O}((m+k)(n+\log m))$ states and $\mathcal{O}(m(n+\log m))$ leaders that computes a given predicate $A\boldsymbol{x} \geq \boldsymbol{c}$, where $A \in \mathbb{Z}^{m\times k}$ and $n$ is the number of bits of the largest entry in $A$ and $\boldsymbol{c}$, in absolute value. Representing $A\boldsymbol{x} \geq \boldsymbol{c}$ as a formula $\varphi$ of QFPA, we obtain a protocol with $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^2)$ leaders that computes $\varphi$. However, in [8] no succinct protocols for formulas with remainder predicates are given, and the paper makes extensive use of leaders.

**Organization.** Sections 2 and 3 introduce basic notation and definitions. Section 4 presents the main result. Sections 5 and 6 present the constructions of the protocols for large and small inputs, respectively. Section 7 presents conclusions. For space reasons, several proofs are only sketched. Detailed proofs are given in the full version of this paper [7].

## 2 Preliminaries

**Notation.** We write $\mathbb{Z}$ to denote the set of integers, $\mathbb{N}$ to denote the set of non negative integers $\{0, 1, \ldots\}$, $[n]$ to denote $\{1, 2, \ldots, n\}$, and $\mathbb{N}^E$ to denote the set of all multisets over $E$, i.e. unordered vectors with components labeled by $E$. The *size* of a multiset $\boldsymbol{v} \in \mathbb{N}^E$ is defined as $|\boldsymbol{v}| \stackrel{\text{def}}{=} \sum_{e\in E} \boldsymbol{v}(e)$. The set of all multisets over $E$ with size $s \geq 0$ is $E^{\langle s\rangle} \stackrel{\text{def}}{=} \{\boldsymbol{v} \in \mathbb{N}^E : |\boldsymbol{v}| = s\}$. We sometimes write multisets using set-like notation, e.g. $\wr a, 2\cdot b \wr$ denotes the multiset $\boldsymbol{v}$ such that $\boldsymbol{v}(a) = 1$, $\boldsymbol{v}(b) = 2$ and $\boldsymbol{v}(e) = 0$ for every $e \in E \setminus \{a, b\}$. The empty multiset $\wr\wr$ is instead denoted $\mathbf{0}$ for readability. For every $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{N}^E$, we write $\boldsymbol{u} \geq \boldsymbol{v}$ if $\boldsymbol{u}(e) \geq \boldsymbol{v}(e)$ for every $e \in E$. Moreover, we write $\boldsymbol{u} + \boldsymbol{v}$ to denote the multiset $\boldsymbol{w} \in \mathbb{N}^E$ such that $\boldsymbol{w}(e) \stackrel{\text{def}}{=} \boldsymbol{u}(e) + \boldsymbol{v}(e)$ for every $e \in E$. The multiset $\boldsymbol{u} \ominus \boldsymbol{v}$ is defined analogously with $-$ instead of $+$, provided that $\boldsymbol{u} \geq \boldsymbol{v}$.

**Presburger arithmetic.** *Presburger arithmetic* (PA) is the first-order theory of $\mathbb{N}$ with addition, i.e. $\mathsf{FO}(\mathbb{N}, +)$. For example, the PA formula $\psi(x, y, z) = \exists x' \exists z' (x = x' + x') \land (y = z + z') \land \neg(z' = 0)$ states that $x$ is even and that $y > z$. It is well-known that for every formula of PA there is an equivalent formula of quantifier-free Presburger arithmetic (QFPA) [13], the theory with syntax given by the grammar

$$\varphi(\boldsymbol{v}) ::= \boldsymbol{a} \cdot \boldsymbol{v} > b \mid \boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b \mid \varphi(\boldsymbol{v}) \wedge \varphi(\boldsymbol{v}) \mid \varphi(\boldsymbol{v}) \vee \varphi(\boldsymbol{v}) \mid \neg\varphi(\boldsymbol{v})$$

where $\boldsymbol{a} \in \mathbb{Z}^X$, $b \in \mathbb{Z}$, $c \in \mathbb{N}_{\geq 2}$, and $\equiv_c$ denotes equality modulo $c$. For example, the formula $\psi(x, y, z)$ above is equivalent to $(x \equiv_2 0) \wedge (y - z \geq 1)$. Throughout the paper, we refer to any formula of QFPA, or the predicate $\mathbb{N}^X \to \{0, 1\}$ it denotes, as a *predicate*. Predicates of the form $\boldsymbol{a} \cdot \boldsymbol{v} > b$ and $\boldsymbol{a} \cdot \boldsymbol{v} \equiv_c b$ are *atomic*, and they are called *threshold* and *remainder* predicates respectively. The *max-norm* $\|\varphi\|$ of a predicate $\varphi$ is the largest absolute value among all coefficients occurring within $\varphi$. The *length* $\text{len}(\varphi)$ of a predicate $\varphi$ is the number of boolean operators occurring within $\varphi$. The *bit length* of a predicate $\varphi$, over variables $X$, is defined as $|\varphi| \overset{\text{def}}{=} \text{len}(\varphi) + \log\|\varphi\| + |X|$. We lift these definitions to sets of predicates in the natural way: given a finite set $P$ of predicates, we define its *size* $\text{size}(P)$ as the number of predicates in $P$, its *length* as $\text{len}(P) \overset{\text{def}}{=} \sum_{\varphi \in P} \text{len}(\varphi)$, its *norm* as $\|P\| \overset{\text{def}}{=} \max\{\|\varphi\| : \varphi \in P\}$, and its *bit length* as $|P| \overset{\text{def}}{=} \text{size}(P) + \text{len}(P) + \log\|P\| + |X|$. Note that $\text{len}(P) = 0$ iff $P$ only contains atomic predicates.

## 3 Population protocols

A *population protocol* is a tuple $\mathcal{P} = (Q, T, L, X, I, O)$ where

- $Q$ is a finite set whose elements are called *states*;
- $T \subseteq \{(\boldsymbol{p}, \boldsymbol{q}) \in \mathbb{N}^Q \times \mathbb{N}^Q : |\boldsymbol{p}| = |\boldsymbol{q}|\}$ is a finite set of *transitions* containing the set $\{(\boldsymbol{p}, \boldsymbol{p}) : \boldsymbol{p} \in \mathbb{N}^Q, |\boldsymbol{p}| = 2\}$;
- $L \in \mathbb{N}^Q$ is the *leader multiset*;
- $X$ is a finite set whose elements are called *input variables*;
- $I : X \to Q$ is the *input mapping*;
- $O : Q \to \{0, 1, \bot\}$ is the *output mapping*.

For readability, we often write $t : \boldsymbol{p} \mapsto \boldsymbol{q}$ to denote a transition $t = (\boldsymbol{p}, \boldsymbol{q})$. Given $\Delta \geq 2$, we say that $t$ is $\Delta$*-way* if $|\boldsymbol{p}| \leq \Delta$.

In the standard syntax of population protocols $T$ is a subset of $\mathbb{N}^2 \times \mathbb{N}^2$, and $O : Q \to \{0, 1\}$. These differences are discussed at the end of this section.

**Inputs and configurations.** An *input* is a multiset $\boldsymbol{v} \in \mathbb{N}^X$ such that $|\boldsymbol{v}| \geq 2$, and a *configuration* is a multiset $C \in \mathbb{N}^Q$ such that $|C| \geq 2$. Intuitively, a configuration represents a population of agents where $C(q)$ denotes the number of agents in state $q$. The *initial configuration* $C_{\boldsymbol{v}}$ *for input* $\boldsymbol{v}$ is defined as $C_{\boldsymbol{v}} \overset{\text{def}}{=} L + \wr \boldsymbol{v}(x) \cdot I(x) : x \in X \wr$.

The *support* and $b$*-support* of a configuration $C$ are respectively defined as $[\![C]\!] \overset{\text{def}}{=} \{q \in Q : C(q) > 0\}$ and $[\![C]\!]_b = \{q \in [\![C]\!] : O(q) = b\}$. The *output* of a configuration $C$ is defined as $O(C) \overset{\text{def}}{=} b$ if $[\![C]\!]_b \neq \emptyset$ and $[\![C]\!]_{\neg b} = \emptyset$ for some $b \in \{0, 1\}$, and $O(C) \overset{\text{def}}{=} \bot$ otherwise. Loosely speaking, if $O(q) = \bot$ then agents in state $q$ have no output, and a population has output $b \in \{0, 1\}$ if all agents *with output* have output $b$.

**Executions.** A transition $t = (\boldsymbol{p}, \boldsymbol{q})$ is *enabled* in a configuration $C$ if $C \geq \boldsymbol{p}$, and *disabled* otherwise. Because of our assumption on $T$, every configuration enables at least one transition. If $t$ is enabled in $C$, then it can be *fired* leading to configuration $C' \overset{\text{def}}{=} C \ominus \boldsymbol{p} + \boldsymbol{q}$, which we denote $C \xrightarrow{t} C'$. For every set of transitions $S$, we write $C \xrightarrow{S} C'$ if $C \xrightarrow{t} C'$ for some $t \in S$.

We denote the reflexive and transitive closure of $\xrightarrow{S}$ by $\xrightarrow{S^*}$. If $S$ is the set of all transitions of the protocol under consideration, then we simply write $\rightarrow$ and $\xrightarrow{*}$.

An execution is a sequence of configurations $\sigma = C_0 C_1 \cdots$ such that $C_i \rightarrow C_{i+1}$ for every $i \in \mathbb{N}$. We write $\sigma_i$ to denote configuration $C_i$. The *output* of an execution $\sigma$ is defined as follows. If there exist $i \in \mathbb{N}$ and $b \in \{0,1\}$ such that $O(\sigma_i) = O(\sigma_{i+1}) = \cdots = b$, then $O(\sigma) \overset{\text{def}}{=} b$, and otherwise $O(\sigma) \overset{\text{def}}{=} \perp$.

**Computations.** An execution $\sigma$ is *fair* if for every configuration $D$ the following holds:

if $|\{i \in \mathbb{N} : \sigma_i \xrightarrow{*} D\}|$ is infinite, then $|\{i \in \mathbb{N} : \sigma_i = D\}|$ is infinite.

In other words, fairness ensures that an execution cannot avoid a configuration forever. We say that a population protocol *computes* a predicate $\varphi \colon \mathbb{N}^X \rightarrow \{0,1\}$ if for every $\boldsymbol{v} \in \mathbb{N}^X$ and every fair execution $\sigma$ starting from $C_{\boldsymbol{v}}$, it is the case that $O(\sigma) = \varphi(\boldsymbol{v})$. Two protocols are *equivalent* if they compute the same predicate. It is known that population protocols compute precisely the Presburger-definable predicates [5, 11].

▶ **Example 1.** Let $\mathcal{P}_n = (Q, T, \mathbf{0}, \{x\}, I, O)$ be the protocol where $Q \overset{\text{def}}{=} \{0, 1, 2, 3, \ldots, 2^n\}$, $I(x) \overset{\text{def}}{=} 1$, $O(a) = 1 \overset{\text{def}}{\iff} a = 2^n$, and $T$ contains a transition, for each $a, b \in Q$, of the form $\lbag a, b \rbag \mapsto \lbag 0, a+b \rbag$ if $a + b < 2^n$, and $\lbag a, b \rbag \mapsto \lbag 2^n, 2^n \rbag$ if $a + b \geq 2^n$. It is readily seen that $\mathcal{P}_n$ computes $\varphi(x) \overset{\text{def}}{=} (x \geq 2^n)$. Intuitively, each agent stores a number, initially 1. When two agents meet, one of them stores the sum of their values and the other one stores 0, with sums capping at $2^n$. Once an agent reaches this cap, all agents eventually get converted to $2^n$.

Now, consider the protocol $\mathcal{P}'_n = (Q', T', \mathbf{0}, \{x\}, I', O')$, where $Q' \overset{\text{def}}{=} \{0, 2^0, 2^1, \ldots, 2^n\}$, $I'(x) \overset{\text{def}}{=} 2^0$, $O'(a) = 1 \overset{\text{def}}{\iff} a = 2^n$, and $T'$ contains a transition for each $0 \leq i < n$ of the form $\lbag 2^i, 2^i \rbag \mapsto \lbag 0, 2^{i+1} \rbag$, and a transition for each $a \in Q'$ of the form $\lbag a, 2^n \rbag \mapsto \lbag 2^n, 2^n \rbag$. Using similar arguments as above, it follows that $\mathcal{P}'_n$ also computes $\varphi$, but more succinctly: While $\mathcal{P}_n$ has $2^n + 1$ states, $\mathcal{P}'_n$ has only $n + 1$ states.

**Types of protocols.** A protocol $\mathcal{P} = (Q, T, L, X, I, O)$ is

- *leaderless* if $|L| = 0$, and *has* $|L|$ *leaders* otherwise;
- $\Delta$-*way* if all its transitions are $\Delta$-way;
- *simple* if there exist $\texttt{f}, \texttt{t} \in Q$ such that $O(\texttt{f}) = 0$, $O(\texttt{t}) = 1$ and $O(q) = \perp$ for every $q \in Q \setminus \{\texttt{f}, \texttt{t}\}$ (i.e., the output is determined by the number of agents in $\texttt{f}$ and $\texttt{t}$.)

Protocols with leaders and leaderless protocols compute the same predicates [5]. Every $\Delta$-way protocol can be transformed into an equivalent 2-way protocol with a polynomial increase in the number of transitions [8]. Finally, every protocol can be transformed into an equivalent simple protocol with a polynomial increase in the number of states [7].

## 4 Main result

The main result of this paper is the following theorem:

▶ **Theorem 2.** *Every predicate $\varphi$ of QFPA can be computed by a leaderless population protocol $\mathcal{P}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states. Moreover, $\mathcal{P}$ can be constructed in polynomial time.*

To prove Theorem 2, we first provide a construction that uses $\ell \in \mathcal{O}(|\varphi|^3)$ leaders. If there are at least $|\boldsymbol{v}| \geq \ell$ input agents $\boldsymbol{v}$ (*large inputs*), we will show how the protocol can be made leaderless by having agents encode both their state and the state of some leader. Otherwise, $|\boldsymbol{v}| < \ell$ (*small inputs*), and we will resort to a special construction, with a single leader, that only works for populations of bounded size. We will show how the leader can be simulated collectively by the agents. Hence, we will construct succinct protocols computing $\varphi$ for large and small inputs, respectively. Formally, we prove:

▶ **Lemma 3.** *Let $\varphi$ be a predicate over variables $X$. There exist $\ell \in \mathcal{O}(|\varphi|^3)$ and leaderless protocols $\mathcal{P}_{\geq \ell}$ and $\mathcal{P}_{<\ell}$ with $\mathcal{O}(\text{poly}(|\varphi|))$ states such that:*

**(a)** *$\mathcal{P}_{\geq \ell}$ computes predicate $(|\boldsymbol{v}| \geq \ell) \rightarrow \varphi(\boldsymbol{v})$, and*

**(b)** *$\mathcal{P}_{<\ell}$ computes predicate $(|\boldsymbol{v}| < \ell) \rightarrow \varphi(\boldsymbol{v})$.*

Theorem 2 follows immediately from the lemma: it suffices to take the conjunction of both protocols, which only yields a quadratic blow-up on the number of states, using the classical product construction [3]. The rest of the paper is dedicated to proving Lemma 3. Parts (a) and (b) are shown in Sections 5 and 6, respectively.

In the remainder of the paper, whenever we claim the existence of some protocol $\mathcal{P}$, we also claim polynomial-time constructibility of $\mathcal{P}$ without mentioning it explicitly.

## 5 Succinct protocols for large populations

We show that, for every predicate $\varphi$, there exists a constant $\ell \in \mathcal{O}(|\varphi|^3)$ and a succinct protocol $\mathcal{P}_{\geq \ell}$ computing $(|\boldsymbol{v}| \geq \ell) \rightarrow \varphi(\boldsymbol{v})$. Throughout this section, we say that $n \in \mathbb{N}$ is *large* if $n \geq \ell$, and that a protocol *computes $\varphi$ for large inputs* if it computes $(|\boldsymbol{v}| \geq \ell) \rightarrow \varphi(\boldsymbol{v})$.

We present the proof in a top-down manner, by means of a chain of statements of the form "$A \leftarrow B$, $B \leftarrow C$, $C \leftarrow D$, and $D$". Roughly speaking, and using notions that will be defined in the forthcoming subsections:

- Section 5.1 introduces protocols with helpers, a special class of protocols with leaders. The section shows: $\varphi$ is computable for large inputs by a succinct leaderless protocol (A), if it is computable for large inputs by a succinct protocol with helpers (B).
- Section 5.2 defines protocols that simultaneously compute a set of predicates. The section proves: (B) holds if the set $P$ of atomic predicates occurring within $\varphi$ is simultaneously computable for large inputs by a succinct protocol with helpers (C).
- Section 5.3 introduces protocols with reversible dynamic initialization. The section shows: (C) holds if each atomic predicate of $P$ is computable for large inputs by a succinct protocol with helpers and reversible dynamic initialization (D).
- Section 5.4 shows that (D) holds by exhibiting succinct protocols with helpers and reversible dynamic initialization that compute atomic predicates for large inputs.

### 5.1 From protocols with helpers to leaderless protocols

Intuitively, a protocol with helpers is a protocol with leaders satisfying an additional property: adding more leaders does not change the predicate computed by the protocol. Formally, let $\mathcal{P} = (Q, T, L, X, I, O)$ be a population protocol computing a predicate $\varphi$. We say that $\mathcal{P}$ is a protocol *with helpers* if for every $L' \succeq L$ the protocol $\mathcal{P}' = (Q, T, L', X, I, O)$ also computes $\varphi$, where $L' \succeq L \overset{\text{def}}{=} \forall q \in Q \colon (L'(q) = L(q) = 0 \vee L'(q) \geq L(q) > 0)$. If $|L| = \ell$, then we say that $\mathcal{P}$ is a protocol *with $\ell$ helpers.*

▶ **Theorem 4.** *Let $\mathcal{P} = (Q, T, L, X, I, O)$ be a $\Delta$-way population protocol with $\ell$-helpers computing some predicate $\varphi$. There exists a 2-way leaderless population protocol with $\mathcal{O}(\ell \cdot |X| + (\Delta \cdot |T| + |Q|)^2)$ states that computes $(|\boldsymbol{v}| \geq \ell) \rightarrow \varphi(\boldsymbol{v})$.*

**Proof sketch.** By [8, Lemma 3], $\mathcal{P}$ can be transformed into a 2-way population protocol (with helpers[2]) computing the same predicate $\varphi$, and with at most $|Q| + 3\Delta \cdot |T|$ states. Thus, we assume $\mathcal{P}$ to be 2-way in the rest of the sketch.

For simplicity, assume $X = \{x\}$ and $L = \llbracket 3 \cdot q, 5 \cdot q' \rrbracket$; that is, $\mathcal{P}$ has 8 helpers, and initially 3 of them are in state $q$, and 5 are in $q'$. We describe a leaderless protocol $\mathcal{P}'$ that simulates $\mathcal{P}$ for every input $\boldsymbol{v}$ such that $|\boldsymbol{v}| \geq |L| = \ell$. Intuitively, $\mathcal{P}'$ runs in two phases:

- In the first phase each agent gets assigned a number between 1 and 8, ensuring that each number is assigned to at least one agent (this is the point at which the condition $|\boldsymbol{v}| \geq \ell$ is needed). At the end of the phase, each agent is in a state of the form $(x, i)$, meaning that the agent initially represented one unit of input for variable $x$, and that it has been assigned number $i$. To achieve this, initially every agent is placed in state $(x, 1)$. Transitions are of the form $\llbracket (x, i), (x, i) \rrbracket \mapsto \llbracket (x, i + 1), (x, i) \rrbracket$ for every $1 \leq i \leq 7$. The transitions guarantee that all but one agent is promoted to $(x, 2)$, all but one to $(x, 3)$, etc. In other words, one agent is "left behind" at each step.
- In the second phase, an agent's state is a multiset: agents in state $(x, i)$ move to state $\llbracket I(x), q \rrbracket$ if $1 \leq i \leq 3$, and to state $\llbracket I(x), q' \rrbracket$ if $4 \leq i \leq 8$. Intuitively, after this move each agent has been assigned two jobs: simultaneously simulate a regular agent of $\mathcal{P}$ starting at state $x$, *and* a helper of $L$ starting at state $q$ or $q'$. Since in the first phase each number is assigned to at least one agent, $\mathcal{P}'$ has at least 3 agents simulating helpers in state $q$, and at least 5 agents simulating helpers in state $q'$. There may be many more helpers, but this is harmless, because, by definition, additional helpers do not change the computed predicate.

  The transitions of $\mathcal{P}'$ are designed according to this double role of the agents of $\mathcal{P}'$. More precisely, for all multisets $\boldsymbol{p}, \boldsymbol{q}, \boldsymbol{p}', \boldsymbol{q}'$ of size two, $\llbracket \boldsymbol{p}, \boldsymbol{q} \rrbracket \mapsto \llbracket \boldsymbol{p}', \boldsymbol{q}' \rrbracket$ is a transition of $\mathcal{P}'$ iff $(\boldsymbol{p} + \boldsymbol{q}) \rightarrow (\boldsymbol{p}' + \boldsymbol{q}')$ in $\mathcal{P}$. ◀

## 5.2 From multi-output protocols to protocols with helpers

A *$k$-output population protocol* is a tuple $\mathcal{Q} = (Q, T, L, X, I, O)$ where $O \colon [k] \times Q \rightarrow \{0, 1, \perp\}$ and $\mathcal{Q}_i \overset{\text{def}}{=} (Q, T, L, X, I, O_i)$ is a population protocol for every $i \in [k]$, where $O_i$ denotes the mapping such that $O_i(q) \overset{\text{def}}{=} O(i, q)$ for every $q \in Q$. Intuitively, since each $\mathcal{Q}_i$ only differs by its output mapping, $\mathcal{Q}$ can be seen as a single population protocol whose executions have $k$ outputs. More formally, $\mathcal{Q}$ *computes* a set of predicates $P = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ if $\mathcal{Q}_i$ computes $\varphi_i$ for every $i \in [k]$. Furthermore, we say that $\mathcal{Q}$ is *simple* if $\mathcal{Q}_i$ is simple for every $i \in [k]$. Whenever the number $k$ is irrelevant, we use the term *multi-output population protocol* instead of $k$-output population protocol.

▶ **Proposition 5.** *Assume that every finite set $A$ of* atomic *predicates is computed by some $|A|$-way multi-output protocol with $\mathcal{O}(|A|^3)$ helpers and states, and $\mathcal{O}(|A|^5)$ transitions. Every QFPA predicate $\varphi$ is computed by some simple $|\varphi|$-way protocol with $\mathcal{O}(|\varphi|^3)$ helpers and states, and $\mathcal{O}(|\varphi|^5)$ transitions.*

**Proof sketch.** Consider a binary tree decomposing the boolean operations of $\varphi$. We design a protocol for $\varphi$ by induction on the height of the tree.

The case where the height is 0, and $\varphi$ is atomic, is trivial. We sketch the induction step for the case where the root is labeled with $\wedge$, that is $\varphi = \varphi_1 \wedge \varphi_2$, the other cases are similar. By induction hypothesis, we have simple protocols $\mathcal{P}_1, \mathcal{P}_2$ computing $\varphi_1, \varphi_2$,

[2] Lemma 3 of [8] deals with leaders and not the more specific case of helpers. Nonetheless, computation under helpers is preserved as the input mapping of $\mathcal{P}$ remains unchanged in the proof of the lemma.

respectively. Let $\mathtt{t}_j, \mathtt{f}_j$ be the output states of $\mathcal{P}_j$ for $j \in \{1, 2\}$ such that $O_j(\mathtt{t}_j) = 1$ and $O_j(\mathtt{f}_j) = 0$. We add two new states $\mathtt{t}, \mathtt{f}$ (the output states of the new protocol) and an additional helper starting in state $\mathtt{f}$. To compute $\varphi_1 \wedge \varphi_2$ we add the following transitions for every $b_1 \in \{\mathtt{t}_1, \mathtt{f}_1\}, b_2 \in \{\mathtt{t}_2, \mathtt{f}_2\}$, and $b \in \{\mathtt{t}, \mathtt{f}\}$: $\Lbag b_1, b_2, b \Rbag \mapsto \Lbag b_1, b_2, \mathtt{t} \Rbag$ if $b_1 = \mathtt{t}_1 \wedge b_2 = \mathtt{t}_2$, and $\Lbag b_1, b_2, b \Rbag \mapsto \Lbag b_1, b_2, \mathtt{f} \Rbag$ otherwise. The additional helper computes the conjunction as desired. ◀

## 5.3 From reversible dynamic initialization to multi-output protocols

Let $P = \{\varphi_1, \ldots, \varphi_k\}$ be a set of $k \geq 2$ atomic predicates of arity $n \geq 1$ over a set $X = \{x_1, \ldots, x_n\}$ of variables. We construct a multi-output protocol $\mathcal{P}$ for $P$ of size $\text{poly}(|\varphi_1| + \cdots + |\varphi_k|)$.

Let $\mathcal{P}_1, \ldots, \mathcal{P}_k$ be protocols for $\varphi_1, \ldots, \varphi_k$. Observe that $\mathcal{P}$ cannot be a "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously. Indeed, the states of such a $\mathcal{P}$ are tuples $(q_1, \ldots, q_k)$ of states of $\mathcal{P}_1, \ldots, \mathcal{P}_k$, and so $\mathcal{P}$ would have exponential size in $k$. Further, $\mathcal{P}$ cannot execute $\mathcal{P}_1, \ldots, \mathcal{P}_k$ asynchronously in parallel, because, given an input $\boldsymbol{x} \in \mathbb{N}^n$, it must dispatch $k \cdot \boldsymbol{x}$ agents ($\boldsymbol{x}$ to the input states of each $\mathcal{P}_j$), but it only has $\boldsymbol{x}$. Such a $\mathcal{P}$ would need $(k-1)|\boldsymbol{x}|$ helpers, which is not possible, because a protocol of size $\text{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ can only use $\text{poly}(|\varphi_1| + \cdots + |\varphi_k|)$ helpers, whatever the input $\boldsymbol{x}$.

The solution is to use a more sophisticated parallel asynchronous computation. Consider two copies of inputs, denoted $\overline{X} = \{\overline{x}_1, \ldots, \overline{x}_n\}$ and $\underline{X} = \{\underline{x}_1, \ldots, \underline{x}_n\}$. For each predicate $\varphi$ over $X$, consider predicate $\tilde{\varphi}$ over $\overline{X} \cup \underline{X}$ satisfying $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$ for every $(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) \in \mathbb{N}^{\overline{X} \cup \underline{X}}$. We obtain $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(\boldsymbol{x})$ whenever $k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}} = \boldsymbol{x}$, e.g. for $\overline{\boldsymbol{x}} := \lfloor \frac{\boldsymbol{x}}{k} \rfloor$ and $\underline{\boldsymbol{x}} := \boldsymbol{x} \bmod k$. With this choice, $\mathcal{P}$ needs to dispatch a total of $k\,(|\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}}|) \leq |\boldsymbol{x}| + n \cdot (k-1)^2$ agents to compute $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. That is, $n \cdot (k-1)^2$ helpers are sufficient to compute $\mathcal{P}$. Formally, we define $\tilde{\varphi}$ in the following way:

$$\text{For } \varphi(\boldsymbol{x}) = \left( \sum_{i=1}^{n} \alpha_i x_i > \beta \right), \text{ we define } \tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) := \left( \sum_{i=1}^{n} (k \cdot \alpha_i)\overline{x}_i + \alpha_i \underline{x}_i > \beta \right)$$

and similarly for modulo predicates. For instance, if $\varphi(x_1, x_2) = 3x_1 - 2x_2 > 6$ and $k = 4$, then $\tilde{\varphi}(\overline{x}_1, \underline{x}_1, \overline{x}_2, \underline{x}_2) = 12\overline{x}_1 + 3\underline{x}_1 - 8\overline{x}_2 - 2\underline{x}_2 > 6$. As required, $\tilde{\varphi}(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}) = \varphi(k\overline{\boldsymbol{x}} + \underline{\boldsymbol{x}})$.

Let us now describe how the protocol $\mathcal{P}$ computes $\tilde{\varphi}_1(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}}), \ldots, \tilde{\varphi}_k(\overline{\boldsymbol{x}}, \underline{\boldsymbol{x}})$. Let $\tilde{\mathcal{P}}_1, \ldots, \tilde{\mathcal{P}}_k$ be protocols computing $\tilde{\varphi}_1, \ldots, \tilde{\varphi}_k$. Let $X = \{\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{n}\}$ be the input states of $\mathcal{P}$, and let $\overline{\mathtt{x}}_1^\mathtt{j}, \ldots, \overline{\mathtt{x}}_\mathtt{n}^\mathtt{j}$ and $\underline{\mathtt{x}}_1^\mathtt{j}, \ldots, \underline{\mathtt{x}}_\mathtt{n}^\mathtt{j}$ be the input states of $\tilde{\mathcal{P}}_j$ for every $1 \leq j \leq k$. Protocol $\mathcal{P}$ repeatedly chooses an index $1 \leq i \leq n$, and executes one of these two actions: (a) take $k$ agents from $\mathtt{x}_\mathtt{i}$, and dispatch them to $\overline{\mathtt{x}}_\mathtt{i}^1, \ldots, \overline{\mathtt{x}}_\mathtt{i}^\mathtt{k}$ (one agent to each state); or (b) take one agent from $\mathtt{x}_\mathtt{i}$ and $(k-1)$ helpers, and dispatch them to $\underline{\mathtt{x}}_\mathtt{i}^1, \ldots, \underline{\mathtt{x}}_\mathtt{i}^\mathtt{k}$. The index and the action are chosen nondeterministically. Notice that if for some input $\mathtt{x}_i$, all $\ell$ agents of $\mathtt{x}_\mathtt{i}$ are dispatched, then $k\overline{\mathtt{x}}_\mathtt{i}^\mathtt{j} + \underline{\mathtt{x}}_\mathtt{i}^\mathtt{j} = \ell$ for all $j$. If all agents of $\mathtt{x}_\mathtt{i}$ are dispatched for every $1 \leq i \leq n$, then we say that the *dispatch is correct.*

The problem is that, because of the nondeterminism, the dispatch may or may not be correct. Assume, e.g., that $k = 5$ and $n = 1$. Consider the input $x_1 = 17$, and assume that $\mathcal{P}$ has $n \cdot (k-1)^2 = 16$ helpers. $\mathcal{P}$ may correctly dispatch $\overline{x}_1 = \lfloor \frac{17}{5} \rfloor = 3$ agents to each of $\overline{\mathtt{x}}_1^1, \ldots, \overline{\mathtt{x}}_5^1$ and $\underline{x}_1 = (17 \bmod 5) = 2$ to each of $\underline{\mathtt{x}}_1^1, \ldots, \underline{\mathtt{x}}_5^1$; this gives a total of $(3+2) \cdot 5 = 25$ agents, consisting of the 17 agents for the input plus 8 helpers. However, it may also wrongly dispatch 2 agents to each of $\overline{\mathtt{x}}_1^1, \ldots, \overline{\mathtt{x}}_5^1$ and 4 agents to each of $\underline{\mathtt{x}}_1^1, \ldots, \underline{\mathtt{x}}_5^1$, with a total of $(2+4) \cdot 5 = 30$ agents, consisting of 14 input agents plus 16 helpers. In the second case, each $\mathcal{P}_j$ wrongly computes $\tilde{\varphi}_j(2, 4) = \varphi_j(2 \cdot 5 + 4) = \varphi_j(14)$, instead of the correct value $\varphi_j(17)$.

To solve this problem we ensure that $\mathcal{P}$ can always recall agents already dispatched to $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$ as long as the dispatch is not yet correct. This allows $\mathcal{P}$ to "try out" dispatches until it dispatches correctly, which eventually happens by fairness. For this we design $\mathcal{P}$ so that (i) the atomic protocols $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$ can work with inputs agents that arrive over time (*dynamic initialization*), and (ii) $\tilde{\mathcal{P}}_1, \dots, \tilde{\mathcal{P}}_k$ can always return to their initial configuration and send agents back to $\mathcal{P}$, unless the dispatch is correct (*reversibility*). To ensure that $\mathcal{P}$ stops redistributing after dispatching a correct distribution, it suffices to replace each reversing transition $\boldsymbol{p} \mapsto \boldsymbol{q}$ by transitions $\boldsymbol{p} + \wr \mathtt{x_i} \wr \mapsto \boldsymbol{q} + \wr \mathtt{x_i} \wr$, one for each $1 \leq i \leq n$: All these transitions become disabled when $\mathtt{x_1}, \dots, \mathtt{x_n}$ are not populated.

**Reversible dynamic initialization.** Let us now formally introduce the class of *protocols with reversible dynamic initialization* that enjoys all properties needed for our construction. A simple protocol with *reversible dynamic initialization* (*RDI-protocol* for short) is a tuple $\mathcal{P} = (Q, T_\infty, T_\dagger, L, X, I, O)$, where $\mathcal{P}_\infty = (Q, T_\infty, L, X, I, O)$ is a simple population protocol, and $T_\dagger$ is the set of transitions making the system reversible, called the *RDI-transitions*.

Let $T \overset{\text{def}}{=} T_\infty \cup T_\dagger$, and let $\mathsf{In} \overset{\text{def}}{=} \{\mathsf{in}_x : x \in X\}$ and $\mathsf{Out} \overset{\text{def}}{=} \{\mathsf{out}_x : x \in X\}$ be the sets of *input* and *output transitions*, respectively, where $\mathsf{in}_x \overset{\text{def}}{=} (\mathbf{0}, \wr I(x) \wr)$ and $\mathsf{out}_x \overset{\text{def}}{=} (\wr I(x) \wr, \mathbf{0})$. An *initialization sequence* is a finite execution $\pi \in (T \cup \mathsf{In} \cup \mathsf{Out})^*$ from the *initial configuration* $L'$ with $L' \succeq L$. The *effective input* of $\pi$ is the vector $\boldsymbol{w}$ such that $\boldsymbol{w}(x) \overset{\text{def}}{=} |\pi|_{\mathsf{in}_x} - |\pi|_{\mathsf{out}_x}$ for every $x \in X$. Intuitively, a RDI-protocol starts with helpers only, and is dynamically initialized via the input and output transitions.

Let $\mathtt{f}, \mathtt{t} \in Q$ be the unique states of $\mathcal{P}$ with $O(\mathtt{f}) = 0$ and $O(\mathtt{t}) = 1$. For every configuration $C$, let $[C] \overset{\text{def}}{=} \{C' : C'(\mathtt{f}) + C'(\mathtt{t}) = C(\mathtt{f}) + C(\mathtt{t})$ and $C'(q) = C(q)$ for all $q \in Q \setminus \{\mathtt{f}, \mathtt{t}\}\}$. Intuitively, all configurations $C' \in [C]$ are equivalent to $C$ in all but the output states.

An RDI-protocol is required to be *reversible*, that is for every initialization sequence $\pi$ with effective input $\boldsymbol{w}$, and such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the following holds:

- if $C \xrightarrow{T^*} D$ and $D' \in [D]$, then $D' \xrightarrow{T^*} C'$ for some $C' \in [C]$, and
- $C(I(x)) \leq \boldsymbol{w}(x)$ for all $x \in X$.

Intuitively, an RDI-protocol can never have more agents in an input state than the effective number of agents it received via the input and output transitions. Further, an RDI-protocol can always reverse all sequences that do not contain input or output transitions. This reversal does not involve the states $\mathtt{f}$ and $\mathtt{t}$, which have a special role as output states. Since RDI-protocols have a default output, we need to ensure that the default output state is populated when dynamic initialization ends, and reversal for $\mathtt{f}$ and $\mathtt{t}$ would prevent that.

An RDI-protocol $\mathcal{P}$ *computes* $\varphi$ if for every initialization sequence $\pi$ with effective input $\boldsymbol{w}$ such that $L' \xrightarrow{\pi} C$ for some $L' \succeq L$, the standard population protocol $\mathcal{P}_\infty$ computes $\varphi(\boldsymbol{w})$ from $C$ (that is with $T_\dagger$ disabled). Intuitively, if the dynamic initialization terminates, the RDI-transitions $T_\dagger$ become disabled, and then the resulting standard protocol $\mathcal{P}_\infty$ converges to the output corresponding to the dynamically initialized input.

▶ **Theorem 6.** *Assume that for every atomic predicate $\varphi$, there exists a $|\varphi|$-way RDI-protocol with $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states and $\mathcal{O}(|\varphi|^3)$ transitions that computes $\varphi$. For every finite set $P$ of atomic predicates, there exists a $|P|$-way simple* multi-output *protocol, with $\mathcal{O}(|P|^3)$ helpers and states, and $\mathcal{O}(|P|^5)$ transitions, that computes $P$.*

## 5.4 Atomic predicates under reversible dynamic initialization

Lastly, we show that atomic predicates are succinctly computable by RDI-protocols:

▶ **Theorem 7.** *Every atomic predicate $\varphi$ over variables $X$ can be computed by a simple $|\varphi|$-way population protocol with reversible dynamic initialization that has $\mathcal{O}(|\varphi|)$ helpers, $\mathcal{O}(|\varphi|^2)$ states, and $\mathcal{O}(|\varphi|^3)$ transitions.*

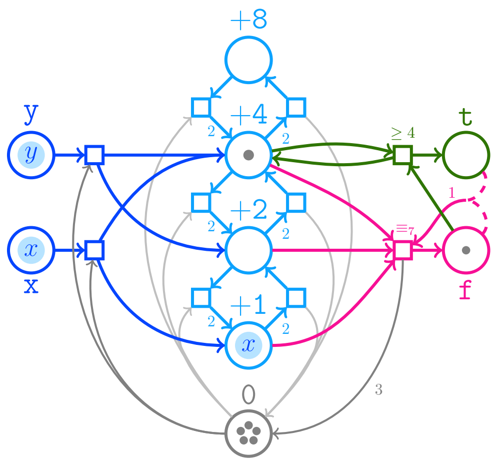


**Figure 1** Partial representation of the protocol computing $5x + 6y \geq 4 \pmod 7$ as a Petri net, where places (circles), transitions (squares) and tokens (smaller filled circles) represent respectively states, transitions and agents. Non-helper agents remember their input variable (labeled here within tokens). The depicted configuration is obtained from input $x = 2$, $y = 1$ by firing the bottom leftmost transition (dark blue).

The protocols for arbitrary threshold and remainder predicates satisfying the conditions of Theorem 7, and their correctness proofs, are given in [7]. Note that the threshold protocol is very similar to the protocol for linear inequalities given in Section 6 of [8]. Thus, as an example, we will instead describe how to handle the remainder predicate $5x - y \equiv_7 4$. Note, that the predicate can be rewritten as $(5x + 6y \geq 4 \pmod 7)) \wedge (5x + 6y \not\geq 5 \pmod 7))$. As we can handle negations and conjunctions separately in Section 5.2, we will now explain the protocol for $\varphi \overset{\text{def}}{=} 5x + 6y \geq 4 \pmod 7$. The protocol is partially depicted in Figure 1 using Petri net conventions for the graphical representation.

The protocol has an *input state* x for each variable $x \in X$, *output states* f and t, a *neutral state* 0, and *numerical states* of the form $+2^{\mathtt{i}}$ for every $0 \leq i \leq n$, where $n$ is the smallest number such that $2^n > \|\varphi\|$. Initially, (at least) one helper is set to f and (at least) $2n$ helpers set to 0. In order to compute $5x + 6y \geq 4 \pmod 7$ for $x := r$ and $y := s$, we initially place $r$ and $s$ agents in the states x and y, i.e., the agents in state x encode the number $r$ in unary, and similarly for y. The blue transitions on the left of Figure 1 "convert" each agents in input states to a binary representation of their corresponding coefficient. In our example, agents in state x are converted to $\boldsymbol{a}(x) = 5 = 0101_2$ by putting one agent in 4 and another one in 1. Since two agents are needed to encode 5, the transition "recruits" one helper from state 0. Observe that, since the inputs can be arbitrarily large, but a protocol can only use a constant number of helpers, the protocol must reuse helpers in order to convert all agents in input states. This happens as follows. If two agents are in the same power of two, say $+2^{\mathtt{i}}$, then one of them can be "promoted" to $+2^{\mathtt{i+1}}$, while the other one moves to state `0`, “liberating” one helper. This allows the agents to represent the overall value of all converted agents in the most efficient representation. That is, from any configuration, one can always reach a configuration where there is at most one agent in each place $2^0, \dots, 2^{n-1}$, there are at most the number of agents converted from input places in place $2^n$, and hence there are at least $n$ agents in place 0, thus ready to convert some agent from the input place. Similar to promotions, “demotions” to smaller powers of two can also happen. Thus, the agents effectively shift through all possible binary representations of the overall value of all converted agents. The $\equiv_7$ transition in Figure 1 allows 3 agents in states `4`, `2` and `1` to “cancel out” by moving to state `0`, and it moves the output helper to `f`. Furthermore, there are RDI-transitions that allow to revert the effects of conversion and cancel transitions. These are not shown in Figure 1.

We have to show that this protocol computes $\varphi$ under reversible dynamic initialization. First note, that while dynamic initialization has not terminated, all transitions have a corresponding reverse transition. Thus, it is always possible to return to wrong initial configurations. However, reversing the conversion transitions can create more agents in input states than the protocol effectively received. To forbid this, each input agent is “tagged” with its variable (see tokens in Figure 1). Therefore, in order to reverse a conversion transitions, the original input agent is needed. This implies, that the protocol is reversible.

Next, we need to argue that the protocol without the RDI-transitions computes $\varphi$ once the dynamic initialization has terminated. The agents will shift through the binary representations of the overall value. Because of fairness, the $\equiv_7$ transition will eventually reduce the overall value to at most 6. There is a $\geq 4$-transition which detects the case where the final value is at least 4 and moves the output helper from `f` to state `t`. Notice that whenever transition $\equiv_7$ occurs, we reset the output by moving the output helper to state `f`.

## 6 Succinct protocols for small populations

We show that for every predicate $\varphi$ and constant $\ell = \mathcal{O}(|\varphi|^3)$, there exists a succinct protocol $\mathcal{P}_{<\ell}$ that computes the predicate $(|\boldsymbol{v}| < \ell) \rightarrow \varphi(\boldsymbol{v})$. In this case, we say that $\mathcal{P}_{<\ell}$ *computes* $\varphi$ *for small inputs*. Further, we say that a number $n \in \mathbb{N}$ (resp. an input $\boldsymbol{v}$) is small with respect to $\varphi$ if $n \leq \ell$ (resp. $|\boldsymbol{v}| \leq \ell$). We present the proof strategy in a top-down manner.

- Section 6.1 proves: There is a succinct leaderless protocol $\mathcal{P}$ that computes $\varphi$ for small inputs (A), if for every small $n$ some succinct protocol $\mathcal{P}_n$ computes $\varphi$ for all inputs of size $n$ (B). Intuitively, constructing a succinct protocol for all small inputs reduces to the simpler problem of constructing a succinct protocol for all small inputs of a fixed size.
- Section 6.2 introduces halting protocols. It shows: There is a succinct protocol that computes $\varphi$ for inputs of size $n$, if for every *atomic* predicate $\psi$ of $\varphi$ some halting succinct protocol computes $\psi$ for inputs of size $n$ (C). Thus, constructing protocols for arbitrary predicates reduces to constructing *halting* protocols for atomic predicates.
- Section 6.3 proves (C). Given a threshold or remainder predicate $\varphi$ and a small $n$, it shows how to construct a succinct halting protocol that computes $\varphi$ for inputs of size $n$.

### 6.1 From fixed-sized protocols with one leader to leaderless protocols

We now define when a population protocol computes a predicate *for inputs of a fixed size*. Intuitively, it should compute the correct value for every initial configurations of this size; for inputs of other sizes, the protocol may converge to the wrong result, or may not converge.

▶ **Definition 8.** *Let $\varphi$ be a predicate and let $i \geq 2$. A protocol $\mathcal{P}$* computes $\varphi$ for inputs of size $i$*, denoted "$\mathcal{P}$* computes $(\varphi \mid i)$*", if for every input $\boldsymbol{v}$ of size $i$, every fair execution of $\mathcal{P}$ starting at $C_{\boldsymbol{v}}$ stabilizes to $\varphi(\boldsymbol{v})$.*

We show that if, for each small number $i$, some succinct protocol computes $(\varphi \mid i)$, then there is a single succinct protocol that computes $\varphi$ for all small inputs.

▶ **Theorem 9.** *Let $\varphi$ be a predicate over a set of variables $X$, and let $\ell \in \mathbb{N}$. Assume that for every $i \in \{2, 3, \ldots, \ell - 1\}$, there exists a protocol with at most one leader and at most $m$ states that computes $(\varphi \mid i)$. Then, there is a leaderless population protocol with $\mathcal{O}(\ell^4 \cdot m^2 \cdot |X|^3)$ states that computes $(\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$.*

**Proof sketch.** Fix a predicate $\varphi$ and $\ell \in \mathbb{N}$. For every $2 \leq i < \ell$, let $\mathcal{P}_i$ be a protocol computing $(\varphi \mid i)$. We describe the protocol $\mathcal{P} = (Q, T, X, I, O)$ that computes $(\boldsymbol{x} \geq \ell) \vee \varphi(\boldsymbol{x}) \equiv (\boldsymbol{x} < \ell) \to \varphi(\boldsymbol{x})$. The input mapping $I$ is the identity. During the computation, agents never forget their initial state – that is, all successor states of an agent are annotated with their initial state. The protocol initially performs a leader election. Each provisional leader stores how many agents it has "knocked out" during the leader election in a counter from 0 to $\ell - 1$. After increasing the counter to a given value $i < \ell$, it resets the state of $i$ agents and itself to the corresponding initial state of $\mathcal{P}_{i+1}$, annotated with $X$, and initiates a simulation of $\mathcal{P}_{i+1}$. When the counter of an agent reaches $\ell - 1$, the agent knows that the population size must be $\geq \ell$, and turns the population into a permanent 1-consensus. Now, if the population size $i$ is smaller than $\ell$, then eventually a leader gets elected who resets the population to the initial population of $\mathcal{P}_i$. Since $\mathcal{P}_i$ computes $(\varphi \mid i)$, the simulation of $\mathcal{P}_i$ eventually yields the correct output. ◀

## 6.2 Computing boolean combinations of predicates for fixed-size inputs

We want to produce a population protocol $\mathcal{P}$ for a boolean combination $\varphi$ of atomic predicates $(\varphi_i)_{i \in [k]}$ for which we have population protocols $(\mathcal{P}_i)_{i \in [k]}$. As in Section 5.3, we cannot use a standard "product protocol" that executes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ synchronously because the number of states would be exponential in $k$. Instead, we want to simulate the *concatenation* of $(\mathcal{P}_i)_{i \in [k]}$. However, this is only possible if for all $i \in [k]$, the executions of $\mathcal{P}_i$ eventually "halt", i.e. some agents are eventually certain that the output of the protocol will not change anymore, which is not the case in general population protocols. For this reason we restrict our attention to "halting" protocols.

▶ **Definition 10.** *Let $\mathcal{P}$ be a simple protocol with output states $\texttt{f}$ and $\texttt{t}$. We say that $\mathcal{P}$ is a* halting protocol *if every configuration $C$ reachable from an initial configuration satisfies:*

- $C(\texttt{f}) = 0 \vee C(\texttt{t}) = 0$,
- $C \xrightarrow{*} C' \wedge C(q) > 0 \Rightarrow C'(q) > 0$ *for every $q \in \{\texttt{f}, \texttt{t}\}$ and every configuration $C'$.*

Intuitively, a halting protocol is a simple protocol in which states $\texttt{f}$ and $\texttt{t}$ behave like "final states": If an agent reaches $q \in \{\texttt{f}, \texttt{t}\}$, then the agent stays in $q$ forever. In other words, the protocol reaches consensus 0 (resp. 1) iff an agent ever reaches $\texttt{f}$ (resp. $\texttt{t}$).

▶ **Theorem 11.** *Let $k, i \in \mathbb{N}$. Let $\varphi$ be a boolean combination of atomic predicates $(\varphi_j)_{j \in [k]}$. Assume that for every $j \in [k]$, there is a simple halting protocol $\mathcal{P}_j = (Q_j, L_j, X, T_j, I_j, O_j)$ with one leader computing $(\varphi_j \mid i)$. Then there exists a simple halting protocol $\mathcal{P}$ that computes $(\varphi \mid i)$, with one leader and $\mathcal{O}\left(|X| \cdot (\mathrm{len}(\varphi) + |Q_1| + \ldots + |Q_k|)\right)$ states.*

**Proof sketch.** We only sketch the construction for $\varphi = \varphi_1 \wedge \varphi_2$. The main intuition is that, since $\mathcal{P}_1$ and $\mathcal{P}_2$ are halting, we can construct a protocol that, given an input $\boldsymbol{v}$, first simulates $\mathcal{P}_1$ on $\boldsymbol{v}$, and, after $\mathcal{P}_1$ halts, either halts if $\mathcal{P}_1$ converges to 0, or simulates $\mathcal{P}_2$ on $\boldsymbol{v}$ if $\mathcal{P}_1$ converges to 1. Each agent remembers in its state the input variable it corresponds to, in order to simulate $\mathcal{P}_2$ on $\boldsymbol{v}$. ◀

## 6.3 Computing atomic predicates for fixed-size inputs

We describe a halting protocol that computes a given threshold predicate for fixed-size inputs.

▶ **Theorem 12.** *Let $\varphi(\boldsymbol{x}, \boldsymbol{y}) \overset{def}{=} \boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$. For every $i \in \mathbb{N}$, there exists a halting protocol with one leader and $\mathcal{O}(i^2(|\varphi| + \log i)^3)$ states that computes $(\varphi \mid i)$.*

We first describe a sequential algorithm Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$, that for every input $\boldsymbol{x}, \boldsymbol{y}$ satisfying $|\boldsymbol{x}| + |\boldsymbol{y}| = i$ decides whether $\boldsymbol{\alpha} \cdot \boldsymbol{x} - \boldsymbol{\beta} \cdot \boldsymbol{y} > 0$ holds. Then we simulate Greater-Sum by means of a halting protocol with $i$ agents.

Since each agent can only have $\mathcal{O}(\log i + \log |\varphi|)$ bits of memory (the logarithm of the number of states), Greater-Sum must use at most $\mathcal{O}(i \cdot (\log i + \log |\varphi|))$ bits of memory, otherwise it cannot be simulated by the agents. Because of this requirement, Greater-Sum cannot just compute, store, and then compare $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ and $\boldsymbol{\beta} \cdot \boldsymbol{y}$; this uses too much memory.

Greater-Sum calls procedures $Probe_1(j)$ and $Probe_2(j)$ that return the $j$-th bits of $\boldsymbol{\alpha x}$ and $\boldsymbol{\beta y}$, respectively, where $j = 1$ is the most significant bit. Since $|\boldsymbol{x}| \leq i$, and the largest constant in $\boldsymbol{\alpha}$ is at most $||\varphi||$, we have $\boldsymbol{\alpha} \cdot \boldsymbol{x} \leq i \cdot ||\varphi||$, and so $\boldsymbol{\alpha} \cdot \boldsymbol{x}$ has at most $m \overset{\text{def}}{=} |\varphi| + \lfloor \log(i) \rfloor + 1$ bits, and the same holds for $\boldsymbol{\beta y}$. So we have $1 \leq j \leq m$. Let us first describe Greater-Sum, and then $Probe_1(j)$; the procedure $Probe_2(j)$ is similar.

Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ loops through $j = 1, \ldots, m$. For each $j$, it calls $Probe_1(j)$ and $Probe_2(j)$. If $Probe_1(j) > Probe_2(j)$, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 1$, otherwise it moves to $j + 1$. If Greater-Sum reaches the end of the loop, then it answers $\varphi(\boldsymbol{x}, \boldsymbol{y}) = 0$. Observe that Greater-Sum only needs to store the current value of $j$ and the bits returned by $Probe_1(j)$ and $Probe_2(j)$. Since $j \leq m$, Greater-Sum only needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

$Probe_1(j)$ uses a decreasing counter $k = m, \ldots, j$ to successively compute the bits $b_1(k)$ of $\boldsymbol{\alpha} \cdot \boldsymbol{x}$, starting at the least significant bit. To compute $b_1(k)$, the procedure stores the carry $c_k \leq i$ of the computation of $b_1(k+1)$; it then computes the sum $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ (where $\boldsymbol{\alpha}(k)$ is the $k$-th vector of bits of $\boldsymbol{\alpha}$), and sets $b_k := s_k \bmod 2$ and $c_{k-1} := s_k \div 2$. The procedure needs $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory for counter $k$, $\log(i) + 1$ bits for encoding $s_k$, and $\mathcal{O}(\log(i))$ bits for encoding $c_k$. So it only uses $\mathcal{O}(\log(|\varphi| + \log i))$ bits of memory.

Let us now simulate Greater-Sum$(\boldsymbol{x}, \boldsymbol{y})$ by a halting protocol with one leader agent. Intuitively, the protocol proceeds in rounds corresponding to the counter $k$. The leader stores in its state the value $j$ and the current values of the program counter, of counter $k$, and of variables $b_k$, $s_k$, and $c_k$. The crucial part is the implementation of the instruction $s_k := c_k + \boldsymbol{\alpha}(k) \cdot \boldsymbol{x}$ of $Probe_1(j)$. In each round, the leader adds input agents one by one. As the protocol only needs to work for populations with $i$ agents, it is possible for each agent to know if it already interacted with the leader in this round, and for the leader to count the number of agents it has interacted with this round, until it reaches $i$ to start the next round.

## 7 Conclusion and further work

We have proved that every predicate $\varphi$ of quantifier-free Presburger arithmetic (QFPA) is computed by a leaderless protocol with poly$(|\varphi|)$ states. Further, the protocol can be computed in polynomial time. The number of states of previous constructions was exponential both in the bit-length of the coefficients of $\varphi$, and in the number of occurrences of boolean connectives. Since QFPA and PA have the same expressive power, every computable predicate has a succinct leaderless protocol. This result completes the work initiated in [8], which also constructed succinct protocols, but only for some predicates, and with the help of leaders.

It is known that protocols with leaders can be exponentially faster than leaderless protocols. Indeed, every QFPA predicate is computed by a protocol with leaders whose expected time to consensus is polylogarithmic in the number of agents [6], while every leaderless protocol for the majority predicate needs at least linear time in the number of agents [1]. Our result shows that, if there is also an exponential gap in state-complexity, then it must be because some family of predicates have protocols with leaders of logarithmic size, while all leaderless families need polynomially many states. The existence of such a family is an open problem.

The question of whether protocols with $\text{poly}(|\varphi|)$ states exist for every PA formula $\varphi$, possibly with quantifiers, also remains open. However, it is easy to prove that no algorithm for the construction of protocols from PA formulas runs in time $2^{p(n)}$ for any polynomial $p$.

▶ **Theorem 13.** *For every polynomial $p$, every algorithm that accepts a formula $\varphi$ of PA as input, and returns a population protocol computing $\varphi$, runs in time $2^{\omega(p(|\varphi|))}$.*

Therefore, if PA also has succinct protocols, then they are very hard to find.

Our succinct protocols for QFPA have slow convergence (in the usual parallel time model, see e.g. [2]), since they often rely on exhaustive exploration of a number of alternatives, until the right one is eventually hit. The question of whether every QFPA predicate has a succinct *and* fast protocol is very challenging, and we leave it open for future research.

## References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579. SIAM, 2017.

2 Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018. doi:10.1145/3289137.3289150.

3 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

5 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *Proc. 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 292–299, 2006. doi:10.1145/1146381.1146425.

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

7 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for Presburger arithmetic, 2019. arXiv:1910.04600.

8 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: On the minimal size of population protocols. In *Proc. 35th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 16:1–16:14, 2018. doi:10.4230/LIPIcs.STACS.2018.16.

9 David Doty and David Soloveichik. Stable leader election in population protocols requires linear time. *Distributed Computing*, 31(4):257–271, 2018.

10 Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bulletin of the EATCS*, 126, 2018.

11 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

12 Christoph Haase. A survival guide to Presburger arithmetic. *SIGLOG News*, 5(3):67–82, 2018.

13 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes rendus du I^er^ Congrès des mathématiciens des pays slaves*, pages 192–201, 1929.

14 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

# E Fast and Succinct Population Protocols for Presburger Arithmetic

This chapter has been published as a **peer-reviewed conference paper**.

Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. "Fast and Succinct Population Protocols for Presburger Arithmetic". In: *1st Symposium on Algorithmic Foundations of Dynamic Networks (SAND 2022)*. Ed. by James Aspnes and Othon Michail. Vol. 221. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 11:1–11:17. DOI: 10.4230/LIPIcs.SAND.2022.11

**Summary.** We present the first synthesis procedure that results in fast and succinct population protocols. Specifically, for a given Presburger formula $\varphi$, we produce a population protocol with agents that have poly($|\varphi|$) states where $|\varphi|$ is the length of the formula with binary coefficients. Further, the expected number of interactions required to reach a stable consensus is $\mathcal{O}(|\varphi|^7 n^2)$, where $n$ is the number of agents. Our synthesis procedure relies on a generalization of population protocols, called population computers, that are easier to design. Specifically, we first construct a succinct population computer for the desired Presburger formula and then transform it into a fast and succinct population protocol.

**Contributions of thesis author.** The author made valuable contributions to the manuscript and played a significant role in the development of the results presented in the paper. In addition, they actively participated in discussions and provided feedback during the revision process. Noteworthy individual contributions are the design and description of the succinct bounded population computers in Section 6, as well as the visualizations of the paper's constructions.

# Fast and Succinct Population Protocols for Presburger Arithmetic

**Philipp Czerner**
Department of Informatics, Technische Universität München, Germany

**Roland Guttenberg**
Department of Informatics, Technische Universität München, Germany

**Martin Helfrich**
Department of Informatics, Technische Universität München, Germany

**Javier Esparza**
Department of Informatics, Technische Universität München, Germany

## Abstract

In their 2006 seminal paper in Distributed Computing, Angluin et al. present a construction that, given any Presburger predicate as input, outputs a leaderless population protocol that decides the predicate. The protocol for a predicate of size $m$ (when expressed as a Boolean combination of threshold and remainder predicates with coefficients in binary) runs in $\mathcal{O}(m \cdot n^2 \log n)$ expected number of interactions, which is almost optimal in $n$, the number of interacting agents. However, the number of states of the protocol is exponential in $m$. This is a problem for natural computing applications, where a state corresponds to a chemical species and it is difficult to implement protocols with many states. Blondin et al. described in STACS 2020 another construction that produces protocols with a polynomial number of states, but exponential expected number of interactions. We present a construction that produces protocols with $\mathcal{O}(m)$ states that run in expected $\mathcal{O}(m^7 \cdot n^2)$ interactions, optimal in $n$, for all inputs of size $\Omega(m)$. For this, we introduce population computers, a carefully crafted generalization of population protocols easier to program, and show that our computers for Presburger predicates can be translated into fast and succinct population protocols.

**2012 ACM Subject Classification** Theory of computation → Distributed computing models

**Keywords and phrases** population protocols, fast, succinct, population computers

**Digital Object Identifier** 10.4230/LIPIcs.SAND.2022.11

**Related Version** *Full Version*: `https://arxiv.org/abs/2202.11601v2` [11]

**Funding** This work was supported by an ERC Advanced Grant (787367: PaVeS) and by the Research Training Network of the Deutsche Forschungsgemeinschaft (DFG) (378803395: ConVeY).

**Acknowledgements** We thank the anonymous reviewers for many helpful remarks. In particular, one remark led to Lemma 11, which in turn led to a nicer formulation of Theorem 2, one of our main results.

## 1 Introduction

Population protocols [4, 5] are a model of computation in which indistinguishable, mobile finite-state agents, randomly interact in pairs to decide whether their initial configuration satisfies a given property, modelled as a predicate on the set of all configurations. The decision is taken by *stable consensus*; eventually all agents agree on whether the property holds or not, and never change their mind again. Population protocols are very close to chemical reaction networks, a model in which agents are molecules and interactions are chemical reactions.

In a seminal paper, Angluin et al. proved that population protocols decide exactly the predicates definable in Presburger arithmetic (PA) [7]. One direction of the result is proved in [5] by means of a construction that takes as input a Presburger predicate and outputs a protocol that decides it. The construction uses the quantifier elimination procedure for PA: every Presburger formula $\varphi$ can be transformed into an equivalent boolean combination of *threshold* predicates of the form $\vec{a} \cdot \vec{x} \geq c$ and *remainder* predicates of the form $\vec{a} \cdot \vec{x} \equiv_m c$, where $\vec{a}$ is an integer vector, $c$ and $m$ are integers, and $\equiv_m$ denotes congruence modulo $m$ [14]. Slightly abusing language, we call the set of these boolean combinations *quantifier-free Presburger arithmetic* (QFPA).[1] Using that PA and QFPA have the same expressive power, Angluin et al. first construct protocols for all threshold and remainder predicates, and then show that the predicates computed by protocols are closed under negation and conjunction.

The two fundamental parameters of a protocol are the expected number of interactions until a stable consensus is reached, and the number of states of each agent. The expected number of interactions divided by the number of agents, also called the parallel execution time, is an adequate measure of the runtime of a protocol when interactions occur in parallel according to a Poisson process [6]. The number of states measures the complexity of an agent. In natural computing applications, where a state corresponds to a chemical species, it is difficult to implement protocols with many states.

Given a formula $\varphi$ of QFPA, let $m$ be the number of bits of the largest coefficient of $\varphi$ in absolute value, and let $s$ be the number of atomic formulas of $\varphi$, respectively. Let $n$ be the number of agents participating in the protocol. The construction of [5] yields a protocol with $\mathcal{O}(s \cdot n^2 \log n)$ expected interactions. Observe that the protocol does not have a leader (an auxiliary agent helping the other agents to coordinate), and agents have a fixed number of states, independent of the size of the population. Under these assumptions, which are also the assumptions of this paper, every protocol for the majority predicate needs $\Omega(n^2)$ expected interactions [1], and so the construction is nearly optimal.[2] However, the number of states is $\Omega(2^{m+s})$, or $\Omega(2^{|\varphi|})$ in terms of the number $|\varphi|$ of bits needed to write $\varphi$ with coefficients in binary. This is well beyond the only known lower bound, showing that for every construction there exist an infinite subset of predicates $\varphi$ for which the construction produces protocols with $\Omega(|\varphi|^{1/4})$ states [9]. So the constructions of [5], and also those of [6, 3, 13], produce *fast* but *very large* protocols.

In [9, 8] Blondin et al. exhibit a construction that produces *succinct* protocols with $\mathcal{O}(\text{poly}(|\varphi|))$ states. However, they do not analyse their stabilisation time. We demonstrate that they run in $\Omega(2^n)$ expected interactions. Loosely speaking, the reason is the use of transitions that "revert" the effect of other transitions. This allows the protocol to "try out" different distributions of agents, retracing its steps until it hits the right one, but also makes it very slow. So [9, 8] produce *succinct* but *very slow* protocols.

Is it possible to produce protocols that are both *fast* and *succinct*? We give an affirmative answer. We present a construction that yields for every formula $\varphi$ of QFPA a protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states and $\mathcal{O}(\text{poly}(|\varphi|) \cdot n^2)$ expected interactions. So our construction achieves optimal stabilisation time in $n$, and, at the same time, yields more succinct protocols than the construction of [8]. Moreover, for inputs of size $\Omega(|\varphi|)$ (a very mild constraint when agents are molecules), we obtain protocols with $\mathcal{O}(|\varphi|)$ states.

[1] Remainder predicates cannot be directly expressed in Presburger arithmetic without quantifiers.

[2] If the model is extended by allowing a *leader* (and one considers the slightly weaker notion of convergence time), or the number of states of an agent is allowed to grow with the population size, $\mathcal{O}(n \cdot \text{polylog}(n))$ interactions can be achieved [6, 3, 2, 13, 12].

Our construction relies on *population computers*, a carefully crafted generalization of the population protocol model of [5]. Population computers extend population protocols in three ways. First, they can exhibit certain *k-way interactions* between more than two agents. Second, they have a more flexible *output condition*, defined by an arbitrary function that assigns an output to every subset of states, instead of to every state.[3] Finally, population computers can use *helpers*: auxiliary agents that, like leaders, help regular agents to coordinate themselves but whose number, contrary to leaders, is not known *a priori*. We exhibit succinct population computers for all Presburger predicates in which every run is finite, and show how to translate such population computers into fast and succinct population protocols.

**Organization of the paper.** We give preliminary definitions in Section 2 and introduce population computers in Section 3. Section 4 gives an overview of the rest of the paper and summarises our main results. Section 5 describes why previous constructions were either not succinct or slow. Section 6 describes population computers for every Presburger predicate. Section 7 converts these computers into succinct population protocols. Section 8 shows that the resulting protocols are also fast.

An extended version of this paper, containing the details of the constructions and all proofs, can be found at [11]. It contains several appendices. Appendix A completes the proofs of Section 5. For the other appendices, there is no one-to-one correspondence to sections of the main paper, instead they are grouped by the construction they analyse. Appendix B concerns the construction of Section 6, but also analyses speed. The four parts of our conversion process are analysed separately, in Appendices C, D, E and F. Appendix G combines the previous to prove the complete conversion theorem. Appendix H summarises the definitions for our speed analyses, and Appendix I contains minor technical lemmata.

## 2 Preliminaries

**Multisets.** Let $E$ be a finite set. A multiset over $E$ is a mapping $E \to \mathbb{N}$, and $\mathbb{N}^E$ denotes the set of all multisets over $E$. We sometimes write multisets using set-like notation, e.g. $\wr a, 2 \cdot b \wr$ denotes the multiset $v$ such that $v(a) = 1$, $v(b) = 2$ and $v(e) = 0$ for every $e \in E \setminus \{a, b\}$. The empty multiset $\wr\wr$ is also denoted $\emptyset$.

For $E' \subseteq E$, $v(E') := \sum_{e \in E'} v(e)$ is the number of elements in $v$ that are in $E'$. The *size* of $v \in \mathbb{N}^E$ is $|v| := v(E)$. The *support* of $v \in \mathbb{N}^E$ is the set $\mathrm{supp}(v) := \{e \in E \mid v(e) > 0\}$. If $E \subseteq \mathbb{Z}$, then we let $\mathrm{sum}(v) := \sum_{e \in E} e \cdot v(e)$ denote the sum of all the elements of $v$. Given $u, v \in \mathbb{N}^E$, $u + v$ and $u - v$ denote the multisets given by $(u + v)(e) := u(e) + v(e)$ and $(u - v)(e) := u(e) - v(e)$ for every $e \in E$. The latter is only defined if $u \geq v$.

**Multiset rewriting transitions.** A *multiset rewriting transition*, or just a *transition*, is a pair $(r, s) \in \mathbb{N}^E \times \mathbb{N}^E$, also written $r \mapsto s$. A transition $t = (r, s)$ is *enabled* at $v \in \mathbb{N}^E$ if $v \geq r$, and its *occurrence* leads to $v' := v - r + s$, denoted $v \to_t v'$. We call $v \to_t v'$ a *step*. The multiset $v$ is *terminal* if it does not enable any transition. An *execution* is a finite or infinite sequence $v_0, v_1, \ldots$ of multisets such that $v \to_{t_1} v_1 \to_{t_2} \cdots$ for some sequence $t_1, t_2, \ldots$ of transitions. A multiset $v'$ is *reachable* from $v$ if there is an execution $v_0, v_1, \ldots, v_k$ with $v_0 = v$ and $v_k = v'$; we also say that the execution *leads from $v$ to $v'$*. An execution is a *run* if it is infinite or it is finite and its last multiset is terminal. A run $v_0, v_1, \ldots$ is *fair* if it is finite, or it is infinite and for every multiset $v$, if $v$ is reachable from $v_i$ for infinitely many $i \geq 0$, then $v = v_j$ for some $j \geq 0$.

[3] Other output conventions for population protocols have been considered [10].

**Presburger arithmetic.** Angluin et al. proved that population protocols decide exactly the predicates $\mathbb{N}^k \to \{0,1\}$ definable in Presburger arithmetic, the first-order theory of addition, which coincide with the *semilinear* predicates [14]. Using the quantifier elimination procedure of Presburger arithmetic, every Presburger predicate can be represented as a Boolean combination of *threshold* and *remainder* predicates. A predicate $\varphi : \mathbb{N}^v \to \{0,1\}$ is a threshold predicate if $\varphi(x_1,...,x_v) = (\sum_{i=1}^{v} a_i x_i \geq c)$, where $a_1,...,a_v, c \in \mathbb{Z}$, and a remainder predicate if $\varphi(x_1,...,x_v) = (\sum_{i=1}^{v} a_i x_i \equiv_m c)$, where $a_1,...,a_v \in \mathbb{Z}$, $m \geq 1$, $c \in \{0,...,m-1\}$, and $a \equiv_m b$ denotes that $a$ is congruent to $b$ modulo $m$. We call the set of these formulas *quantifier-free Presburger arithmetic*, or QFPA. The *size* of a predicate is the minimal number of bits of a formula of QFPA representing it, with coefficients written in binary.

## 3 Population Computers

Population computers are a generalization of population protocols that allows us to give very concise descriptions of our protocols for Presburger predicates.

**Syntax.** A *population computer* is a tuple $\mathcal{P} = (Q, \delta, I, O, H)$, where:

- $Q$ is a finite set of *states*. Multisets over $Q$ are called *configurations*.
- $\delta \subseteq \mathbb{N}^Q \times \mathbb{N}^Q$ is a set of multiset rewriting transitions $r \mapsto s$ over $Q$ such that $|r| = |s| \geq 2$ and $|\text{supp}(r)| \leq 2$. Further, we require that $\delta$ is a partial function, so $s_1 = s_2$ for all $r, s_1, s_2$ with $(r_1 \mapsto s_1), (r_2 \mapsto s_2) \in \delta$. A transition $r \mapsto s$ is *binary* if $|r| = 2$. We call a population computer is *binary* if every transition binary.
- $I \subseteq Q$ is a set of *input states*. An *input* is a configuration $C$ such that $\text{supp}(C) \subseteq \text{supp}(I)$.
- $O : 2^Q \to \{0,1,\perp\}$ is an *output function*. The *output* of a configuration $C$ is $O(\text{supp}(C))$. An output function $O$ is a *consensus output* if there is a partition $Q = Q_0 \cup Q_1$ of $Q$ such that $O(Q') = 0$ iff $Q' \subseteq Q_0$, $O(Q') = 1$ iff $Q' \subseteq Q_1$, and $O(Q') = \perp$ otherwise.
- $H \in \mathbb{N}^{Q \setminus I}$ is a multiset of *helper agents* or just *helpers*. A *helper configuration* is a configuration $C$ such that $\text{supp}(C) \subseteq \text{supp}(H)$ and $C \geq H$.

**Graphical notation.** We visualise population computers as Petri nets (see e.g. Figure 3). Places (circles) and transitions (squares) represent respectively states and transitions. To visualise configurations, we draw agents as tokens (smaller filled circles).

**Semantics.** Intuitively, a population computer decides which output (0 or 1) corresponds to an input $C_I$ as follows. It adds to the agents of $C_I$ an arbitrary helper configuration $C_H$ of agents to produce the initial configuration $C_I + C_H$. Then it starts the computation and lets it stabilise to configurations of output 1 or output 0. Formally, the *initial configurations* of $\mathcal{P}$ for input $C_I$ are all configurations of the form $C_I + C_H$ for some helper configuration $C_H$. A run $C_0 C_1 ...$ *stabilises to* $b$ if there exists an $i \geq 0$ such that $O(\text{supp}(C_i)) = b$ and $C_i$ only reaches configurations $C'$ with $O(\text{supp}(C')) = b$. An input $C_I$ *has output* $b$ if for every initial configuration $C_0 = C_I + C_H$, every fair run starting at $C_0$ stabilises to $b$. A population computer $\mathcal{P}$ *decides* a predicate $\varphi : \mathbb{N}^I \to \{0,1\}$ if every input $C_I$ has output $\varphi(C_I)$.

**Terminating and bounded computers.** A population computer is *bounded* if no run starting at an initial configuration $C$ is infinite, and *terminating* if no fair run starting at $C$ is infinite. Observe that bounded population computers are terminating.

**Size and adjusted size.** Let $\mathcal{P} = (Q, \delta, I, O, H)$ be a population computer. We assume that $O$ is described as a boolean circuit with size$(O)$ gates. For every transition $t = (r \mapsto s)$ let $|t| := |r|$. The *size* of $\mathcal{P}$ is $\text{size}(\mathcal{P}) := |Q| + |H| + \text{size}(O) + \sum_{t\in\delta}|t|$. If $\mathcal{P}$ is binary, then (as for population protocols) we do not count the transitions and define the *adjusted size* $\text{size}_2(\mathcal{P}) := |Q| + |H| + \text{size}(O)$. Observe that both the size of a transition and the size of the helper multiset are the number of elements, i.e. the size in unary, strengthening our later result about the existence of succinct population computers.

**Population protocols.** A population computer $\mathcal{P} = (Q, \delta, I, O, H)$ is a *population protocol* if it is binary, has no helpers ($H = \emptyset$), and $O$ is a consensus output. It is easy to see that this definition coincides with the one of [5]. The speed of a binary population computer with no helpers, and so in particular of a population protocol, is defined as follows. We assume a probabilistic execution model in which at configuration $C$ two agents are picked uniformly at random and execute a transition, if possible, moving to a configuration $C'$ (by assumption they enable at most one transition). This is called an *interaction.* Repeating this process, we generate a *random execution* $C_0C_1...$ . We say that the execution *stabilises* at time $t$ if $C_t$ reaches only configurations $C'$ with $O(\text{supp}(C')) = O(\text{supp}(C_t))$, and we say that $\mathcal{P}$ *decides* $\varphi$ *within* $T$ *interactions* if it decides $\varphi$ and $\mathbb{E}(t) \leq T$. See e.g. [6] for more details.

**Population computers vs. population protocols.** Population computers generalise population protocols in three ways:

- They have non-binary transitions, but only those in which the interacting agents populate at most two states. For example, $\wr p, p, q \wr \mapsto \wr p, q, o \wr$ (which in the following is written simply as $p, p, q \mapsto p, q, o$) is allowed, but $p, q, o \mapsto p, p, q$ is not.
- They use a multiset $H$ of auxiliary helper agents, but the addition of more helpers does not change the output of the computation. Intuitively, contrary to the case of leaders, agents do not know any upper bound on the number of helpers, and so the protocol cannot rely on this bound for correctness or speed.
- They have a more flexible output condition. Loosely speaking, population computers accept by stabilising the population to an accepting set of states, instead of to a set of accepting states.

## 4 Overview and Main Results

Given a predicate $\varphi \in QFPA$ over variables $x_1, ..., x_v$, the rest of this paper shows how to construct a fast and succinct population protocol deciding $\varphi$. First, Section 5 gives an overview of previous constructions and explains why they are not fast or not succinct. Then we proceed in five steps:

1. Construct the predicate $\text{double}(\varphi) \in QFPA$ over variables $x_1, ..., x_v, x'_1, ..., x'_v$ by syntactically replacing every occurrence of $x_i$ in $\varphi$ by $x_i + 2x'_i$. For example, if $\varphi = (x - y \geq 0)$ then $\text{double}(\varphi) = (x + 2x' - y - 2y' \geq 0)$. Observe that $|\text{double}(\varphi)| \in \mathcal{O}(|\varphi|)$.
2. Construct a succinct bounded population computer $\mathcal{P}$ deciding double$(\varphi)$.
3. Convert $\mathcal{P}$ into a succinct population protocol $\mathcal{P}'$ deciding $\varphi$ for inputs of size $\Omega(|\varphi|)$.
4. Prove that $\mathcal{P}'$ runs within $\mathcal{O}(n^3)$ interactions.
5. Use a refined running-time analysis to prove that $\mathcal{P}'$ runs within $\mathcal{O}(n^2)$ interactions.

Section 6 constructs bounded population computers for all predicates $\varphi \in QFPA$. This allows us to conduct steps 1 and 2. More precisely, the section proves:

▶ **Theorem 1.** *For every predicate $\varphi \in$ QFPA there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides $\varphi$.*

Section 7 proves the following conversion theorem (steps 3 and 4).

▶ **Theorem 2.** *Every bounded population computer of size $m$ deciding* double$(\varphi)$ *can be converted into a terminating population protocol with $\mathcal{O}(m^2)$ states which decides $\varphi$ in at most $\mathcal{O}(f(m)\, n^3)$ interactions for inputs of size $\Omega(m)$, for some function $f$.*

Section 8 introduces $\alpha$*-rapid* population computers, where $\alpha \geq 1$ is a certain parameter, and uses a more detailed analysis to show that the population protocols of Theorem 2 are in fact smaller and faster (step 5):

▶ **Theorem 3.**
**(a)** *The population computers constructed in Theorem 1 are $\mathcal{O}(|\varphi|^3)$-rapid.*
**(b)** *Every $\alpha$-rapid population computer of size $m$ deciding* double$(\varphi)$ *can be converted into a terminating population protocol with $\mathcal{O}(m)$ states that decides $\varphi$ in $\mathcal{O}(\alpha\, m^4 n^2)$ interactions for inputs of size $\Omega(m)$.*

The restriction to inputs of size $\Omega(m)$ is very mild. Moreover, it can be lifted using a technique of [8], at the price of adding additional states (and at no cost regarding asymptotic speed, since the speed of the new protocol only changes for inputs of size $\mathcal{O}(m)$):

▶ **Corollary 4.** *For every $\varphi \in$ QFPA there exists a terminating population protocol with $\mathcal{O}(\text{poly}(|\varphi|))$ states that decides $\varphi$ in $\mathcal{O}(f(|\varphi|)\, n^2)$ interactions, for a function $f$.*

It is known that the majority predicate can only be decided in $\Omega(n^2)$ interactions by population protocols [1], so — as a general construction — our result is optimal w.r.t. time. Regarding space, an $\Omega(|\varphi|^{1/4})$ lower bound was shown in [9], leaving a polynomial gap.

## 5 Previous Constructions: Angluin et al. and Blondin et al.

The population protocols for a quantifier free Presburger predicate $\varphi$ constructed in [5] are not *succinct*, i.e. do not have $\mathcal{O}(|\varphi|^a)$ states for any constant $a$, and those of [8] are not *fast*, i.e. do not have speed $\mathcal{O}(|\varphi|^a n^b)$ for any constants $a, b$. We explain why with the help of some examples.

▶ **Example 5.** Consider the protocol of [5] for the predicate $\varphi = (x - y \geq 2^d)$. The states are the triples $(\ell, b, u)$ where $\ell \in \{A, P\}$, $b \in \{Y, N\}$ and $-2^d \leq u \leq 2^d$. Intuitively, $\ell$ indicates whether the agent is active (A) or passive (P), $b$ indicates whether it currently believes that $\varphi$ holds (Y) or not (N), and $u$ is the agent's wealth, which can be negative. Agents for input $x$ are initially in state $(A, N, 1)$, and agents for $y$ in $(A, N, -1)$. If two passive agents meet their encounter has no effect. If at least one agent is active, then the result of the encounter is given by the transition $(*, *, u), (*, *, u') \mapsto (A, b, q), (P, b, r)$ where $b = Y$ if $u + u' \geq 2^d$ else $N$; $q = \max(-2^d, \min(2^d, u + u'))$; and $r = (u + u') - q$. The protocol stabilises after $\mathcal{O}(n^2 \log n)$ expected interactions [5], but it has $2^{d+1} + 1$ states, exponentially many in $|\varphi| \in \Theta(d)$.

▶ **Example 6.** We give a protocol for $\varphi = (x - y \geq 2^d)$ with a polynomial number of states. This is essentially the protocol of [8]. We remove states and transitions from the protocol of Example 5, retaining only the states $(\ell, b, u)$ such that $u$ is a power of 2, and some of the transitions involving these states:

$$\begin{array}{rcll}
(*,*,2^i),(*,*,2^i) & \mapsto & (A,N,2^{i+1}),(P,N,0) & \text{for every } 0 \le i \le d-2 \\
(*,*,2^{d-1}),(*,*,2^{d-1}) & \mapsto & (A,Y,2^d),(P,Y,0) & \\
(*,*,-2^i),(*,*,-2^i) & \mapsto & (A,N,-2^{i+1}),(P,N,0) & \text{for every } 0 \le i \le d-1 \\
(*,*,2^i),(*,*,-2^i) & \mapsto & (A,N,0),(P,N,0) & \text{for every } 0 \le i \le d-1
\end{array}$$

The protocol is not yet correct. For example, for $d = 1$ and the input $x = 2, y = 1$, the protocol can reach in one step the configuration in which the three agents (two $x$-agents and one $y$-agent) are in states $(A,Y,2),(P,Y,0),(A,N,-1)$, after which it gets stuck. In [8] this is solved by adding "reverse" transitions:

$$\begin{array}{rcll}
(A,N,2^{i+1}),(P,N,0) & \mapsto & (A,N,2^i),(P,N,2^i) & \text{for every } 0 \le i \le d-2 \\
(A,Y,2^d),(P,Y,0) & \mapsto & (A,N,2^{d-1}),(P,N,2^{d-1}) & \\
(A,N,-2^{i+1}),(P,N,0) & \mapsto & (A,N,-2^i),(A,N,-2^i) & \text{for every } 0 \le i \le d-1
\end{array}$$

The protocol has only $\Theta(d)$ states and transitions, but runs within $\Omega(n^{2^d-2})$ interactions. Consider the inputs $x, y$ such that $x - y = 2^d$, and let $n := x + y$. Say that an agent is *positive* at a configuration if it has positive wealth at it. The protocol can only stabilise if it reaches a configuration with exactly one positive agent with wealth $2^d$. Consider a configuration with $i < 2^d$ positive agents. The next configuration can have $i-1$, $i$, or $i+1$ positive agents. The probability of $i+1$ positive agents is $\Omega(1/n)$, while that of $i-1$ positive agents is only $\mathcal{O}(1/n^2)$, and the expected number of interactions needed to go from $2^d$ positive agents to only 1 is $\Omega(n^{2^d-1})$ [11, Appendix A.1].

▶ **Example 7.** Given protocols $\mathcal{P}_1, \mathcal{P}_2$ with $n_1$ and $n_2$ states deciding predicates $\varphi_1$ and $\varphi_2$, Angluin et al. construct in [5] a protocol $\mathcal{P}$ for $\varphi_1 \wedge \varphi_2$ with $n_1 \cdot n_2$ states. It follows that the number of states of a protocol for $\varphi := \varphi_1 \wedge \cdots \wedge \varphi_s$ grows exponentially in $s$, and so in $|\varphi|$. Blondin et al. give an alternative construction with polynomially many states [8, Section 5.3]. However, their construction contains transitions that, as in the previous example, reverse the effect of other transitions, and make the protocol very slow. The problem is already observed in the toy protocol with states $q_1, q_2$ and transitions $q_1, q_1 \mapsto q_2, q_2$ and $q_1, q_2 \mapsto q_1, q_1$. (Similar transitions are used in the initialisation of [8].) Starting with an even number $n \ge 2$ of agents in $q_1$, eventually all agents move to $q_2$ and stay there, but the expected number of interactions is $\Omega(2^{n/10})$ [11, Appendix A.2].

## 6 Succinct Bounded Population Computers for Presburger Predicates

In Sections 6.1 and 6.2 we construct population computers for remainder and threshold predicates in which all coefficients are powers of two. We present the remainder case in detail, and sketch the threshold case. The generalization to arbitrary coefficients is achieved by means of a gadget very similar to the one we used to compute boolean combinations of predicates. This later gadget is presented in Section 6.3, and so we introduce the generalization there.

### 6.1 Population computers for remainder predicates

Let $Pow^+ = \{2^i \mid i \ge 0\}$ be the set of positive powers of 2.

We construct population computers $\mathcal{P}_\varphi$ for remainder predicates $\varphi := \sum_{i=1}^{v} a_i x_i \equiv_m c$, where $a_i \in Pow^+ \cap \{0, ..., m-1\}$ for every $1 \le i \le v$, $m \in \mathbb{N}$, and $c \in \{0, ..., m-1\}$. We say that a finite multiset $r$ over $Pow^+$ *represents* the residue $\text{rep}(r) := \text{sum}(r) \bmod m$. For example, if $m = 11$ then $r_{18} := \wr 2^3, 2^3, 2^1 \wr$ represents 7. Accordingly, we call the multisets over $Pow^+$ *representations.* A *representation of degree $d$* only contains elements of $Pow_d^+ := \{2^d, 2^{d-1}, ..., 2^0\}$. A representation $r$ is a *support representation* if $r(x) \leq 1$ for every $x \in Pow^+$; so its represented value is completely determined by the support. For example, $r_{18}$ is not a support representation of 7, but $\wr 2^5, 2^3 \wr$ is.

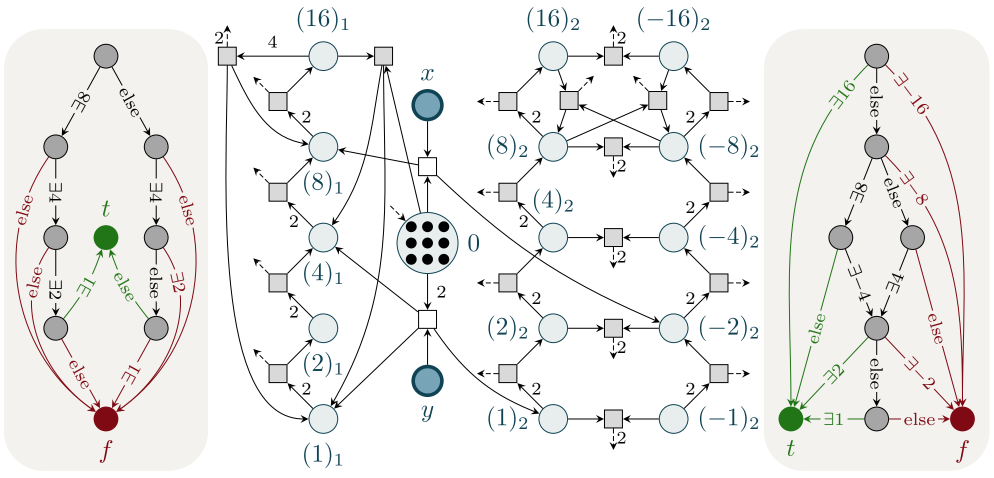


**Figure 1** (middle) Graphical Petri net representation (see Section 3) of population computer for the predicate $\varphi_1 \vee \varphi_2$ with $\varphi_1 = (8x + 5y \equiv_{11} 4)$ and $\varphi_2 = (y - 2x \geq 5)$. All dashed arrows implicitly lead to the reservoir state 0. It has 22 helpers although only 9 are drawn for space reasons. (left) decision diagram for output function of remainder predicate $8x + 5y \equiv_{11} 4$. It checks if the total value is 15 or 4. Starting at the top node of the diagram: if state 8 is populated, we move to the left child, otherwise to the right child; at the left child, if state 4 is populated we move to the right child, etc. (right) decision diagram for output function of threshold predicate $y - 2x \geq 5$.

We proceed to construct $\mathcal{P}_\varphi$. Let us give some intuition first. $\mathcal{P}_\varphi$ has $Pow_d^+ \cup \{0\}$ as set of states. We extend the notion of representation to configurations by disregarding agents in state 0; a configuration is therefore a support representation if all states except 0 have at most one agent. The initial states of $\mathcal{P}_\varphi$ are chosen so that every initial configuration for an input $(x_1, ..., x_v)$ is a representation of the residue $z := \sum_{i=1}^{v} a_i x_i \bmod m$. The transitions transform this initial representation of $z$ into a support representation of $z$. Whether $z \equiv_m c$ holds or not depends only on the support of this representation, and the output function thus returns 1 for the supports satisfying $z \equiv_m c$, and 0 otherwise. Let us now formally describe $\mathcal{P}_\varphi$ for $\varphi := \sum_{i=1}^{v} a_i x_i \equiv_m c$ where $a_i \in Pow^+ \cap \{0, ..., m-1\}$.

**States and initial states.** Let $d := \lceil \log_2 m \rceil$. The set of states is $Q = Pow_d^+ \cup \{0\}$. The set of initial states is $I := \{a_1, ..., a_v\}$. Observe that an input $C_I = \wr x_1 \cdot a_1, ..., x_v \cdot a_v \wr$ is a representation of $z$, but not necessarily a support representation.

**Transitions.** Transitions ensure that non-support representations, i.e. representations with two or more agents in some state $q$, are transformed into representations of the same residue "closer" to a support representation. For $q \in 2^0, ..., 2^{d-1}$ we introduce the transition:

$$2^i, 2^i \quad \mapsto \quad 2^{i+1}, 0 \qquad \text{for } 0 \leq i \leq d-1 \qquad \langle\text{combine}\rangle$$

For $q = 2^d$ we introduce a transition that replaces an agent in $2^d$ by a multiset of agents $r$ with $\mathrm{sum}(r) = 2^d - m$, preserving the residue. Let $b_d b_{d-1} ... b_0$ be the binary encoding of $2^d - m$, and let $\{i_1, ..., i_j\}$ be the positions such that $b_{i_1} = \cdots = b_{i_j} = 1$. The transition is:

$$2^d, 0, ..., 0 \quad \mapsto \quad 2^{i_1}, ..., 2^{i_j} \qquad \langle \mathsf{modulo} \rangle$$

These transitions are enough, but we also add a transition that takes $d$ agents in $2^d$ and replaces them by agents with sum $d \cdot 2^d \bmod m$. Intuitively, this makes the protocol faster. Let $b_d b_{d-1} ... b_0$ and $\{i_1, ..., i_j\}$ be as above, but for $d \cdot 2^d \bmod m$ instead of $2^d - m$.

$$2^d, ..., 2^d \quad \mapsto \quad 2^{i_1}, ..., 2^{i_j}, 0, ..., 0 \qquad \langle \mathsf{fast\ modulo} \rangle$$

**Helpers.** We set $H := \wr 3d \cdot 0 \wr$, i.e. the computer initially places at least $3d$ helper agents in state 0. This makes sure one can always execute the next ⟨modulo⟩ or ⟨fast modulo⟩ transition: if no more agents can be combined, there are at most $d$ agents in the states $2^0, ..., 2^{d-1}$. Thus, there are at least $2d$ agents in the states 0 and $2^d$, enabling one of these transitions. Observe that for every initial configuration $C_I + C_H$ we have $\mathrm{sum}(C_I + C_H) = \mathrm{sum}(C_I)$, and so, abusing language, every initial configuration for $C_I$ is also a representation of $z$.

**Output function.** The computer eventually reaches a support configuration with at most one agent in every state except for 0. Thus, for every support set $S \subseteq Q$, we define $O(S) := 1$ if $\mathrm{sum}(S) \equiv_m c$, and $O(S) = 0$ else. We show the existence of a small boolean circuit for the output function $O$ in the proof of Lemma 8; this can be found in [11, Appendix B.1].

▶ **Lemma 8.** *Let $\varphi := \sum_{i=1}^{v} a_i x_i \equiv_m c$, where $a_i \in \{2^{d-1}, ..., 2^1, 2^0\}$ for every $1 \leq i \leq v$ and $c \in \{0, ..., m-1\}$ with $d := \lceil \log_2 m \rceil$. There is a bounded computer of size $\mathcal{O}(d)$ deciding $\varphi$.*

The left half of Figure 1 shows the population computer for $\varphi = (8x + 5y \equiv_{11} 4)$.

## 6.2 Population computers for threshold predicates

We sketch the construction of population computers $\mathcal{P}_\varphi$ for threshold predicates $\varphi := \sum_{i=1}^{v} a_i x_i \geq c$, where $a_i \in \{2^j, 2^{-j} \mid j \geq 0\}$ for every $1 \leq i \leq v$ and $c \in \mathbb{N}$. As the construction is similar to the construction for remainder, we will focus on the differences and refer to [11, Appendix B.2] for details.

As for remainder, we work with representations that are multisets of powers of 2. However, they represent the sum of their elements (without modulo) and we allow both positive and negative powers of 2. Similar to the remainder construction, the computer transforms any representation into a *support representation* without changing the represented value. Then, the computer decides the predicate using only the support of that representation.

Again, there are ⟨combine⟩ transitions that allow agents with the same value to combine. Instead of modulo transitions, ⟨cancel⟩ transitions further simplify the representation: $2^i, -2^i \mapsto 0, 0$. Note that even after exhaustively applying ⟨combine⟩ and ⟨cancel⟩ there can still be many agents in $2^d$ or many agents in $-2^d$. This has two consequences:

- In the construction for general predicates of Section 6.3, we need that computers for remainder and threshold move most agents to state 0. In the remainder construction, all but a constant number of agents are moved to 0. In contrast, the threshold construction does not have this property. Thus, we do not design a single computer for a given threshold predicate $\varphi$ but a family: one for every degree $d$ larger than some minimum degree $d_0 \in \Omega(|\varphi|)$. Intuitively, larger degrees result in a larger fraction of agents in 0.

- Assume we detect agents in $2^d$ ($-2^d$ is analogous). If there are many, the predicate is true. However, if there is just one, then the represented value might be small, due to negative contributions $-2^0, ..., -2^{d-1}$. We cannot distinguish the two cases, so we add transition ⟨cancel 2nd highest⟩: $2^d, -2^{d-1} \mapsto 2^{d-1}, 0$. It ensures that agents cannot be present in both $2^d$ and $-2^{d-1}$; therefore, an agent in $2^d$ certifies a value of at least $2^{d-1}$.

The right half of Figure 1 shows the population computer for $\varphi = (-2x + y \geq 5)$ with degree $d = 4$. [11, Appendix B.2] proves:

▶ **Lemma 9.** *Let $\varphi := \sum_{i=1}^{v} a_i x_i \geq c$, where $a_i \in \{2^j, 2^{-j} \mid j \geq 0\}$ for every $1 \leq i \leq v$. For every $d \geq \max\{\lceil \log_2 c \rceil + 1, \lceil \log_2 |a_1| \rceil, ..., \lceil \log_2 |a_v| \rceil\}$ there is a bounded computer of size $\mathcal{O}(d)$ that decides $\varphi$.*

## 6.3 Population computers for all Presburger predicates

We present a construction that, given threshold or remainder predicates $\varphi_1, ..., \varphi_s$, yields a population computer $\mathcal{P}$ deciding an arbitrary given boolean combination $B(\varphi_1, ..., \varphi_s)$ of $\varphi_1, ..., \varphi_s$. We only sketch the construction, see [11, Appendix B.3] for details. We use the example $\varphi_1 = (y - 2x \geq 5)$, $\varphi_2 = (8x + 5y \equiv_{11} 4)$ and $B(\varphi_1, \varphi_2) = \varphi_1 \vee \varphi_2$. The result of the construction for this example is shown in Figure 1. The construction has 6 steps:

**1. Rewrite Predicates.** The constructions in Sections 6.1 and 6.2 only work for predicates where all coefficients are powers of 2. We transform each predicate $\varphi_i$ into a new predicate $\varphi'_i$ where all coefficients are decomposed into their powers of 2. In our example, $\varphi'_1 := \varphi_1$ because all coefficients are already powers of 2. However, $\varphi_2(x, y) = (8x + 5y \equiv_{11} 4)$ is rewritten as $\varphi'_2(x, y_1, y_2) := (8x + 4y_1 + 1y_2 \equiv_{11} 4)$ because $5 = 4 + 1$. Note that $\varphi_2(x, y) = \varphi'_2(x, y, y)$ holds for every $x, y \in \mathbb{N}$. Let $r$ be the size of the largest split of a coefficient, i.e. $r = 2$ in the example.

**2. Construct Subcomputers.** For every $1 \leq i \leq s$, if $\varphi_i$ is a remainder predicate, then let $\mathcal{P}_i$ be the computer defined in Section 6.1. If $\varphi_i$ is a threshold predicate, then let $\mathcal{P}_i$ be the computer of Section 6.2, with $d = d_0 + \lceil \log_2 s \rceil$. We explain this choice of $d$ in step 5.

**3. Combine Subcomputers.** Take the disjoint union of $\mathcal{P}_i$, but merging their 0 states. More precisely, rename all states $q \in Q_i$ to $(q)_i$, with the exception of state 0. Construct a computer with the union of all the renamed states and transitions. Figure 1 shows the Petri net representation of the computer so obtained for our example. We call the combined 0 state *reservoir* as it holds agents with no value that are needed for various tasks like input distribution.

**4. Input Distribution.** For each variable $x_i$ add a corresponding new input state $x_i$. Then add a transition that takes an agent in state $x_i$ and agents in 0 and distributes agents to the input states of the subcomputers that correspond to $x_i$. In our example, we add two states $x$ and $y$ and the transitions $x, 0 \mapsto (1)_1, (8)_2$ and $y, 0, 0 \mapsto (-2)_1, (4)_2, (1)_2$. The distribution for $x$ needs one helper, because we need one agent in each subcomputer. The distribution for $y$ needs two helpers, one for $\mathcal{P}_1$ and two for $\mathcal{P}_2$, as $5y$ was split into $4y_1 + 1y_2$. This way, once the input states are empty, the correct value is distributed to each subcomputer. Crucially, this input distribution can be fast as it is not reversible.

**5. Add Extra Helpers.** In addition to all helpers from the subcomputers, add $r-1$ more helpers to state 0. Intuitively, this allows to distribute the first input agent. Because of our choice for $d$ in threshold subcomputers, each subcomputer returns most agents back to state 0. More precisely, for each distribution the number of agents that do not get returned to 0 only increases by at most $\frac{1}{s}$ (per subcomputer). So in total only one agent is "consumed" per distribution and enough agents are returned to 0 for the next distribution to occur. In our example, the agents that stay in each of the $s = 2$ subcomputers only increases by at most $\frac{1}{2}$ per distribution. (In fact, remainder subcomputers return all distributed agents.)

**6. Combine Output.** Note that we can still decide $\varphi_i$ from the support of the states in the corresponding subcomputer $\mathcal{P}_i$. We compute the output for $\varphi$ by combining the outputs of the subcomputers $\mathcal{P}_1, ..., \mathcal{P}_s$ according to $B(\varphi_1, ..., \varphi_s)$. In our example, we set the output to 1 if and only if the output of $\mathcal{P}_1$ or $\mathcal{P}_2$ is 1.

In [11, Appendix B.3], we show that this computer is succinct, correct and bounded:

▶ **Theorem 1.** *For every predicate $\varphi \in$ QFPA there exists a bounded population computer of size $\mathcal{O}(|\varphi|)$ that decides $\varphi$.*

## 7 Converting Population Computers to Population Protocols

In this section we prove Theorem 2. We proceed in four steps, which must be carried out in the given order. Section 7.1 converts any bounded computer $\mathcal{P}$ for double($\varphi$) of size $m$ into a *binary* bounded computer $\mathcal{P}_1$ with $\mathcal{O}(m^2)$ states. Section 7.2 converts $\mathcal{P}_1$ into a binary bounded computer $\mathcal{P}_2$ with a *marked consensus output function* (a notion defined in the section). Section 7.3 converts $\mathcal{P}_2$ into a binary bounded computer $\mathcal{P}_3$ for $\varphi$ — not double($\varphi$) — with a marked consensus output function *and no helpers*. Section 7.4 shows that $\mathcal{P}_3$ runs within $\mathcal{O}(n^3)$ interactions. Finally, we convert $\mathcal{P}_3$ to a binary terminating (not necessarily bounded) computer $\mathcal{P}_4$ with a *normal consensus output* and no helpers, also running within $\mathcal{O}(n^3)$ interactions. This uses standard ideas; for space reasons it is described only in the full version at [11, Appendix F]. Similarly, the other conversions and results are only sketched, with details in [11].

### 7.1 Removing multiway transitions

We transform a bounded population computer with $k$-way transitions $r \mapsto s$ such that $|\text{supp}(r)| \leq 2$ into a binary bounded population computer. Let us first explain why the construction introduced in [9, Lemma 3], which works for arbitrary transitions $r \mapsto s$, is too slow. In [9], the 3-way transition $t : q_1, q_2, q_3 \mapsto q_1', q_2', q_3'$ is simulated by the transitions

$$t_1 : q_1, q_2 \mapsto w, q_{12} \qquad t_2 : q_{12}, q_3 \mapsto c_{12}, q_3' \qquad t_3 : q_3', w \mapsto q_1', q_2' \qquad \bar{t}_1 : w, q_{12} \mapsto q_1, q_2$$

Intuitively, the occurrence of $t_1$ indicates that two agents in $q_1$ and $q_2$ want to execute $t$, and are waiting for an agent in $q_3$. If the agent arrives, then all three execute $t_2 t_3$, which takes them to $q_1', q_2', q_3'$. Otherwise, the two agents must be able to return to $q_1, q_2$ to possibly execute other transitions. This is achieved by the "revert" transition $\bar{t}_1$. The construction for a $k$-way transition has "revert" transitions $\bar{t}_1, ..., \bar{t}_{k-2}$. As in Example 6 and Example 7, these transitions make the final protocol very slow.

We present a gadget without "revert" transitions that works for $k$-way transitions $r \mapsto s$ satisfying $|\text{supp}(r)| \leq 2$. Figure 2 illustrates it, using Petri net notation, for the 5-way transition $t : \wr 3p, 2q \wr \mapsto \wr a, b, c, d, e \wr$. In the gadget, states $p$ and $q$ are split into $(p, 0), ..., (p, 3)$

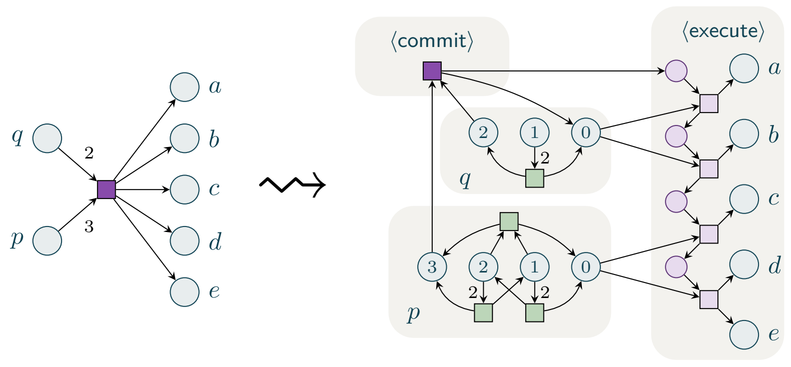


**Figure 2** Simulating the 5-way transition $\Lbag 3 \cdot p, 2 \cdot q \mapsto a, b, c, d, e \Rbag$ by binary transitions.

and $(q, 0), ..., (q, 2)$. Intuitively, an agent in $(q, i)$ acts as representative for a group of $i$ agents in state $q$. Agents in $(p, 3)$ and $(q, 2)$ commit to executing $t$ by executing the binary transition ⟨commit⟩. After committing, they move to the states $a, ..., e$ together with the other members of the group, who are "waiting" in the states $(p, 0)$ and $(q, 0)$. Note that ⟨commit⟩ is binary because of the restriction $|\text{supp}(r)| \leq 2$ for multiway transitions.

To ensure correctness of the conversion, agents can commit to transitions if they represent more than the required amount. In this case, the initiating agents would commit to a transition and then elect representatives for the superfluous agents, before executing the transition. This requires additional intermediate states.

[11, Appendix C] formalises the gadget and proves its correctness and speed.

## 7.2 Converting output functions to marked-consensus output functions

We convert a computer with an arbitrary output function into another one with a *marked-consensus* output function. An output function is a *marked-consensus* output function if there are disjoint sets of states $Q_0, Q_1 \subseteq Q$ such that $O(S) := b$ if $S \cap Q_b \neq \emptyset$ and $S \cap Q_{1-b} = \emptyset$, for $b \in \{0, 1\}$, and $O(S) := \bot$ otherwise. Intuitively, for every $S \subseteq Q$ we have $O(S) = 1$ if all agents agree to avoid $Q_0$ (consensus), and at least one agent populates $Q_1$ (marked consensus). We only sketch the construction, a detailed description as well as a graphical example can be found in [11, Appendix D].

Our starting point is some bounded and binary computer $\mathcal{P} = (Q, \delta, I, O, H)$, e.g. as constructed in Section 7.1. Let $(G, E)$ be a boolean circuit with only NAND-gates computing the output function $O$. We simulate $\mathcal{P}$ by a computer $\mathcal{P}'$ with a marked consensus output and $\mathcal{O}(|Q| + |G|)$ states. This result allows us to bound the number of states of $\mathcal{P}'$ by applying well known results on the complexity of Boolean functions.

Intuitively, $\mathcal{P}'$ consists of two processes running asynchronously in parallel. The first one is (essentially, see below) the computer $\mathcal{P}$ itself. The second one is a gadget that simulates the execution of $G$ on the support of the current configuration of $\mathcal{P}$. Whenever $\mathcal{P}$ executes a transition, it raises a flag indicating that the gadget must be reset (for this, we duplicate each state $q \in Q$ into two states $(q, +)$ and $(q, -)$, indicating whether the flag is raised or lowered). Crucially, $\mathcal{P}$ is bounded, and so it eventually performs a transition *for the last time*. This resets the gadget for the last time, after which the gadget simulates $(G, E)$ on the support of the terminal configuration reached by $\mathcal{P}$.

The gadget is designed to be operated by one *state-helper* for each $q \in Q$, with set of states $Q_{\text{supp}}(q)$, and a *gate-helper* for each gate $g \in G$, with set of states $Q_{\text{gate}}(g)$, defined as follows:

- $Q_{\text{supp}}(q) := \{q\} \times \{0, 1, !\}$. These states indicate that $q$ belongs/does not belong to the support of the current configuration (states $(q, 0)$ and $(q, 1)$), or that the output has changed from 0 to 1 (state $(q, !)$).
- $Q_{\text{gate}}(g) := \{g\} \times \{0, 1, \perp\}^3$ for each gate $g \in G$, storing the current values of the two inputs of the gate and its output. Uninitialised values are stored as $\perp$.

Recall that a population computer must also remain correct for a larger number of helpers. This is ensured by letting all helpers populating one of these sets, say $Q_{\text{supp}}(q)$, perform a leader election; whenever two helpers in states of $Q_{\text{supp}}(q)$ meet, one of them becomes a non-leader, and a flag requesting a complete reset of the gadget is raised. All resets are carried out by a *reset-helper* with set of states $Q_{\text{reset}} := \{0, ..., |Q| + |G|\}$, initially in state 0. (Reset-helpers also carry out their own leader election!) Whenever a reset is triggered, the reset-helper contacts all other $|Q| + |G|$ helpers in round-robin fashion, asking them to reset the computation.

Eventually the original protocol $\mathcal{P}$ has already reached a terminal configuration with some support $Q_{\text{term}}$, each set $Q_{\text{supp}}(q)$ and $Q_{\text{gate}}(g)$ is populated by exactly one helper, and all previous resets are terminated. From this moment on, $\mathcal{P}$ never changes its configuration. The $|Q|$ state-helpers detect the support $Q_{\text{term}}$ of the terminal configuration by means of transitions that move them to the states $Q_{\text{term}} \times \{1\}$ and $(Q \setminus Q_{\text{term}}) \times \{0\}$; the gate-helpers execute $(G, E)$ on input $Q'$ by means of transitions that move them to the states describing the correct inputs and outputs for each gate. State-helpers use $Q \times \{!\}$ as intermediate states, indicating that the circuit must recompute its output.

It remains to choose the sets $Q_0$ and $Q_1$ of states the marked consensus output. We do it according to the output $b$ of the output gate $g_{\text{out}} \in G$: $Q_b$ is the set of states of $Q_{\text{gate}}(g_{\text{out}})$ corresponding to output $b$.

## 7.3 Removing helpers

We convert a bounded binary computer $\mathcal{P}$ deciding the predicate double$(\varphi)$ over variables $x_1, ..., x_k, x'_1, ..., x'_k$ into a computer $\mathcal{P}'$ with no helpers deciding $\varphi$ over variables $x_1, ..., x_k$. In [8], a protocol with helpers and set of states $Q$ is converted into a protocol without helpers with states $Q \times Q$. We sketch a better construction that avoids the quadratic blowup. A detailed description can be found in [11, Appendix E].

Let us give some intuition first. All agents of an initial configuration of $\mathcal{P}'$ are in input states. $\mathcal{P}'$ simulates $\mathcal{P}$ by *liberating* some of these agents and transforming them into helpers, without changing the output of the computation. For this, two agents in an input state $x_i$ are allowed to interact, producing one agent in $x'_i$ and one "liberated" agent, which can be used as a helper. This does not change the output of the computation, because double$(\varphi)(..., x_i, ..., x'_i, ...) =$ double$(\varphi)(..., x_i - 2, ..., x'_i + 1, ...)$ holds by definition of double$(\varphi)$.

Figure 3 illustrates this idea. Assume $\mathcal{P}$ has input states $x, y, x', y'$ and helpers $H = \wr q_1, q_2, q_3, q_4 \wr$, as shown on the left-hand side. Assume further that $\mathcal{P}$ computes a predicate double$(\varphi)(x, y, x', y')$. The computer $\mathcal{P}'$ is shown on the right of the figure. The additional transitions liberate agents, and send them to the helper states $H$. Observe that the initial states of $\mathcal{P}'$ are only $x$ and $y$. Let us see why $\mathcal{P}'$ decides $\varphi(x, y)$. As the initial configuration of

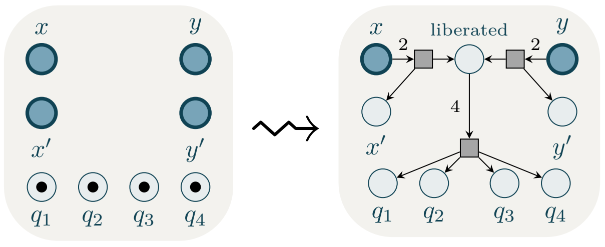


**Figure 3** Illustration in graphical Petri net notation (see Section 3) of construction that removes helpers. Initial states are highlighted.

$\mathcal{P}'$ for an input $x, y$ puts no agents in $x', y'$, the computer $\mathcal{P}'$ produces the same output on input $x, y$ as $\mathcal{P}$ on input $x, y, 0, 0$. Since $\mathcal{P}$ decides double($\varphi$) and double($\varphi$)$(x, y, 0, 0) = \varphi(x, y)$ by the definition of double($\varphi$), we are done. We make some remarks:

- $\mathcal{P}'$ may liberate more agents than necessary to simulate the multiset $H$ of helpers of $\mathcal{P}$. This is not an issue, because by definition additional helpers do not change the output of the computation.
- If the input is too small, $\mathcal{P}'$ cannot liberate enough agents to simulate $H$. Therefore, the new computer only works for inputs of size $\Omega(|H|) = \Omega(|\varphi|)$.
- Even if the input is large enough, $\mathcal{P}'$ might move agents out of input states before liberating enough helpers. However, the computers of Section 6 can only do this if there are enough helpers in the reservoir state (see point 3. in Section 6.3). Therefore, they always generate enough helpers when the input is large enough.

## 7.4 A $\mathcal{O}(n^3)$ bound on the expected interactions

We show that the computer obtained after the previous conversion runs within $\mathcal{O}(n^3)$ interactions. We sketch the main ideas; the details are in [11, Appendix G].

We introduce *potential functions* that assign to every configuration a positive *potential*, with the property that executing any transition strictly decreases the potential. Intuitively, every transition "makes progres". We then prove two results: (1) under a mild condition, a computer has a potential function iff it is bounded, and (2) every binary computer with a potential function and no helpers, i.e. any bounded computer for which speed is defined, stabilises within $\mathcal{O}(n^3)$ interactions. This concludes the proof.

Fix a population computer $\mathcal{P} = (Q, \delta, I, O, H)$.

▶ **Definition 10.** *A function $\Phi : \mathbb{N}^Q \to \mathbb{N}$ is* linear *if there exist weights $w : Q \to \mathbb{N}$ s.t. $\Phi(C) = \sum_{q \in Q} w(q)C(q)$ for every $C \in \mathbb{N}^Q$. We write $\Phi(q)$ instead of $w(q)$. A* potential function *(for $\mathcal{P}$) is a linear function $\Phi$ such that $\Phi(r) \geq \Phi(s) + |r| - 1$ for all $(r \mapsto s) \in \delta$.*

Observe that $k$-way transitions reduce the potential by $k - 1$, binary transitions by 1. At this point, we consider only binary computers, but this distinction becomes relevant for the refined speed analysis.

If a population computer has a potential function, then every run executes at most $\mathcal{O}(n)$ transitions, and so the computer is bounded. Applying Farkas' Lemma we can show that the converse holds for computers in which every state can be populated – a mild condition, since states that can never be populated can be deleted without changing the behaviour.

▶ **Lemma 11.** *If $\mathcal{P}$ has a reachable configuration $C_q$ with $C_q(q) > 0$ for each $q \in Q$, then $\mathcal{P}$ is bounded iff there is a potential function for $\mathcal{P}$.*

Consider now a binary computer with a potential function and no helpers. At every non-terminal configuration, at least one (binary) transition is enabled. The probability that two agents chosen uniformly at random enable this transition is $\Omega(1/n^2)$, and so a transition occurs within $\mathcal{O}(n^2)$ interactions. Since the computer has a potential function, every run executes at most $\mathcal{O}(n)$ transitions, and so the computer stabilises within $\mathcal{O}(n^3)$ interactions.

The final step to produce a population protocol is to translate computers with marked-consensus output function into computers with standard consensus output function, while preserving the number of interactions. For space reasons this construction is presented in [11, Appendix F].

## 8 Rapid Population Computers: Proving a $\mathcal{O}(n^2)$ Bound

We refine our running-time analysis to show that the population protocols we have constructed actually stabilise within $\mathcal{O}(n^2)$ interactions. We continue to use potential functions, as introduced in Section 7.4, but improve our analysis as follows:

- We introduce *rapidly-decreasing* potential functions. Intuitively, their existence shows that progress is not only *possible*, but also *likely*. We prove that they certify stabilisation within $\mathcal{O}(n^2)$ interactions.
- We introduce *rapid* population computers, as computers with rapidly-decreasing potential functions that also satisfy some technical conditions. We convert rapid computers into protocols with $O(|\varphi|)$ states, and show that the computers of Section 6 are rapid.

In order to define rapidly-decreasing potential functions, we need a notion of "probability to execute a transition" that generalises to multiway transitions and is preserved by our conversions. At a configuration $C$ of a protocol, the probability of executing a binary transition $t = (p, q \mapsto p', q')$ is $C(q)C(p)/n(n-1)$. Intuitively, leaving out the normalisation factor $1/n(n-1)$, the transition has "speed" $C(q)C(p)$, proportional in the *product* of the number of agents in $p$ and $q$. But for a multiway transition like $q, q, p \mapsto r_1, r_2, r_3$ the situation changes. If $C(q) = 2$, it does not matter how many agents are in $p$ – the transition is always going to take $\Omega(n^2)$ interactions. We therefore define the speed of a transition as $\min\{C(q), C(p)\}^2$ instead of $C(q)C(p)$.

For the remainder of this section, let $\mathcal{P} = (Q, \delta, I, O, H)$ denote a population computer.

▶ **Definition 12.** *Given a configuration* $C \in \mathbb{N}^Q$ *and some transition* $t = (r \mapsto s) \in \delta$*, we let* $\text{tmin}_t(C) := \min\{C(q) : q \in \text{supp}(r)\}$*. For a set of transitions* $T \subseteq \delta$*, we define* $\text{speed}_T(C) := \sum_{t \in T} \text{tmin}_t(C)^2$*, and write* $\text{speed}(C) := \text{speed}_\delta(C)$ *for convenience.*

▶ **Definition 13.** *Let* $\Phi$ *denote a potential function for* $\mathcal{P}$ *and let* $\alpha \geq 1$*. We say that* $\Phi$ *is* $\alpha$-rapidly decreasing *at a configuration* $C$ *if* $\text{speed}(C) \geq (\Phi(C) - \Phi(C_{\text{term}}))^2/\alpha$ *for all terminal configurations* $C_{\text{term}}$ *with* $C \to C_{\text{term}}$*.*

We have not been able to find potential functions for the computers of Section 6 that are rapidly decreasing at every reachable configuration, only at reachable configurations with sufficiently many helpers, defined below. Fortunately, that is enough for our purposes.

▶ **Definition 14.** $C \in \mathbb{N}^Q$ *is* well-initialised *if* $C$ *is reachable and* $C(I) + |H| \leq \frac{2}{3}n$*.*

Observe that an initial configuration $C$ can only be well-initialised if $C(\text{supp}(H)) \in \Omega(C(I))$. We now define *rapid* population computers, and state the result of our improved analysis.

▶ **Definition 15.** *$\mathcal{P}$ is* $\alpha$-rapid *if*

1. *it has a potential function $\Phi$ which is $\alpha$-rapidly decreasing in well-initialised configurations,*
2. *every state of $\mathcal{P}$ but one has at most 2 outgoing transitions,*
3. *all configurations in $\mathbb{N}^I$ are terminal, and*
4. *for all transitions $t = (r \mapsto s)$, $q \in I$ we have $r(q) \leq 1$ and $s(q) = 0$.*

▶ **Theorem 3.**

**(a)** *The population computers constructed in Theorem 1 are $\mathcal{O}(|\varphi|^3)$-rapid.*

**(b)** *Every $\alpha$-rapid population computer of size $m$ deciding* $\mathrm{double}(\varphi)$ *can be converted into a terminating population protocol with $\mathcal{O}(m)$ states that decides $\varphi$ in $\mathcal{O}(\alpha\, m^4 n^2)$ interactions for inputs of size $\Omega(m)$.*

The detailed proofs can be found in the full version [11], in the following sections. The proof of (a) is given in Appendix B. For (b), we prove separate theorems for each conversion in Appendices C, D, E, and F. To achieve a tighter analysis of our conversions, we generalise the notion of potential function; this is described in Appendix H.

## 9 Conclusions

We have shown that every predicate $\varphi$ of quantifier-free Presburger arithmetic has a population protocol with $\mathcal{O}(\mathrm{poly}(|\varphi|))$ states and $\mathcal{O}(|\varphi|^7 \cdot n^2)$ expected number of interactions. If only inputs of size $\Omega(|\varphi|)$ matter, we give a protocol with $\mathcal{O}(|\varphi|)$ states and the same speed. The obvious point for further improvement is the $|\varphi|^7$ factor in the expected number of interactions.

Our construction is close to optimal. Indeed, for every construction there is an infinite family of predicates for which it yields protocols with $\Omega(|\varphi|^{1/4})$ states [9]; further, it is known that every protocol for the majority predicate requires in $\Omega(n^2)$ interactions.

### References

1 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-space trade-offs in population protocols. In *SODA*, pages 2560–2579. SIAM, 2017.

2 Dan Alistarh and Rati Gelashvili. Recent algorithmic advances in population protocols. *SIGACT News*, 49(3):63–73, 2018.

3 Dan Alistarh, Rati Gelashvili, and Milan Vojnovic. Fast and exact majority in population protocols. In *PODC*, pages 47–56. ACM, 2015.

4 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC*, pages 290–299. ACM, 2004.

5 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Comput.*, 18(4):235–253, 2006.

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

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

8 Michael Blondin, Javier Esparza, Blaise Genest, Martin Helfrich, and Stefan Jaax. Succinct population protocols for Presburger arithmetic. In *STACS*, volume 154 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

9 Michael Blondin, Javier Esparza, and Stefan Jaax. Large flocks of small birds: On the minimal size of population protocols. In *STACS*, volume 96 of *LIPIcs*, pages 16:1–16:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

**10** Robert Brijder, David Doty, and David Soloveichik. Democratic, existential, and consensus-based output conventions in stable computation by chemical reaction networks. *Natural Computing*, 17(1):97–108, 2018.

**11** Philipp Czerner, Roland Guttenberg, Martin Helfrich, and Javier Esparza. Fast and succinct population protocols for Presburger arithmetic, 2022. `arXiv:2202.11601v2`.

**12** David Doty, Mahsa Eftekhari, Leszek Gasieniec, Eric E. Severson, Grzegorz Stachowiak, and Przemyslaw Uznanski. Brief announcement: A time and space optimal stable population protocol solving exact majority. In *PODC*, pages 77–80. ACM, 2021.

**13** Robert Elsässer and Tomasz Radzik. Recent results in population protocols for exact majority and leader election. *Bull. EATCS*, 126, 2018.

**14** Christoph Haase. A survival guide to Presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018.