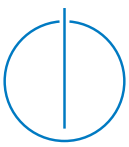
DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Automatic Verification of non-silent Population Protocols

Martin Helfrich



DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Automatic Verification of non-silent Population Protocols

Automatische Verifikation von non-silent Population Protocols

Author:	Martin Helfrich
Supervisor:	Univ.-Prof. Dr. Dr. h.c. Javier Esparza
Advisor:	Philipp Meyer
Submission date:	16.08.2019

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, 16.08.2019 Martin Helfrich

Abstract

Population protocols are a model of distributed computation by anonymous, identical, finite-state agents. A population protocol computes a predicate if all fair executions starting in some initial configuration converge to the lasting consensus that is determined by the predicate. Deciding if a protocol computes a given predicate has non-elementary complexity. In an earlier work Blondin *et al.* have given an algorithm to automatically verify *silent* population protocols that has lower (DP) complexity.

In this work we introduce *stage graphs*, a structure describing all possible fair executions from all possible initial configurations of a population protocol. We show that stage graphs can be used to automatically verify silent and non-silent protocols.

We present an incomplete constraint-based algorithm that automatically computes a stage graph for a given protocol. If successful, the resulting stage graph is a certificate for the correctness of the protocol. This approach can verify all silent protocols tested by Blondin *et al.* as well as many other non-silent protocols from the literature.

Additionally, our algorithm can compute state-of-the-art upper bounds for the expected number of interactions needed until a protocol terminates. Furthermore, we show how to verify linear temporal logic (LTL) specifications like liveness.

Contents

1. Introduction

Population protocols [1][2] are a model of distributed computation. They describe systems of identical and anonymous agents with a finite number of states. These agents are passively mobile and interact with each other. Prominent examples of such systems are sensor networks and chemical reaction networks. Structurally, population protocols are closely related to Petri nets and vector addition systems [3].

The global state of a population protocol is called a *configuration*. Since all agents are identical and anonymous, a configuration is fully determined by the number of agents in each local state. In each step of the computation, a group of agents is chosen fairly. They interact and change their states according to some joint transition function.

A protocol computes a boolean value for a given initial configuration if all agents eventually agree to this value, i.e. if they reach a lasting consensus. The predicate computed by a protocol is the function that assigns to each initial configuration C the boolean value computed by the protocol starting from C.

The correctness problem for population protocols answers the question if a given protocol computes a given predicate. For a specific input, the semantics of a protocol is a finite graph. However, there are infinitely many possible inputs resulting in an infinite state-space. Therefore, one cannot simply use a model checker to automatically verify a population protocol for all inputs. In [3] Esparza *et al.* show that the correctness problem is at least as hard as the reachability problem for Petri nets. Thus, the correctness problem is at least *TOWER*-hard [4]. This implies non-elementary complexity.

In [5] Blondin *et al.* describe a method that allows automatic verification of *silent* population protocols. This is the subclass of population protocols where eventually the configuration does not change anymore. In other words, every computation eventually reaches a terminal configuration. Thus, to verify a silent protocol, it is enough to show that every terminal configuration is a consensus with the correct output.

Even though there is a silent population protocol for every predicate [2], many protocols discussed in the literature are not silent (e.g. [6][7]). Those protocols do not reach a terminal configuration. Instead, they reach a lasting consensus where the configuration keeps on changing. This more complicated termination behaviour is not as easy to characterize when compared to silent protocols. Thus, we need a different approach if we want to verify silent and non-silent population protocols.

In this work we verify population protocols using *stage graphs*. A stage graph

describes infinitely many possible executions of a population protocol by grouping configurations into stages. Those stages are chosen such that executions entering a stage cannot exit the stage (inductivity) but will eventually enter some substage. As stage graphs do not contain cycles, any execution starting in some stage of the graph will eventually reach a stage without successors. We will show that stage graphs can act as certificates for properties of population protocols such as correctness.

Given a (Presburger) precondition φ_{pre} and a (Presburger) postcondition φ_{post} we give a construction for a φ_{pre}-φ_{post}-stage-graph. If the construction is successful, the resulting stage graph verifies that any execution starting in a configuration where φ_{pre} holds will eventually reach a point where φ_{post} holds forever. The constructed stages correspond to non-reversible changes to the structure of the configurations like:

- *dead transitions*: A transition is dead if it can never occur again.
- *deserted states*: A state is deserted if it can never be populated again.

In practice, we are not only interested in the correctness of a population protocol but also the speed of the protocol. This speed is defined as the expected number of interactions needed to reach a lasting consensus when the next pair of interacting agents is chosen uniformly at random. In [8] Blondin *et al.* show how to automatically compute an upper bound for the speed of a protocol. We extend our construction so that we compute such upper bounds while computing a stage graph.

Population protocols can be used to model a wide range of distributed or concurrent systems. For example, they can be used to describe mutex algorithms. In such cases, we want to verify properties like liveness (e.g. transition t can always be fired again) that cannot be verified using the given construction. Such properties can be expressed using *linear temporal logic* (LTL). Given a population protocol and an LTL formula, we show how to construct a different population protocol, such that we can verify the formula using a stage graph.

Structure of this work. In Section 2 we introduce population protocols as well as a modified version of LTL that is used to describe fair executions. In Section 3 we introduce stage graphs and show that they can verify properties of population protocols. The construction that computes a φ_{pre}-φ_{post}-stage-graph is explained in Section 4. In Section 5 we extend the construction to compute the expected number of interactions until termination. Section 6 shows the construction to verify transition-based LTL formulas. We test our implementation on many protocols from the literature and report the results in Section 7. Finally, we conclude in Section 8.

2. Preliminaries

In this section we recall the definition of Presburger arithmetic and define population protocols. Furthermore, we introduce two example protocols as well as version of linear temporal logic.

Presburger arithmetic. Presburger arithmetic is the first order logic over integers. In the usual definition predicates can be defined using the constants "0" and "1", the functions "+" and "<", the logical operands "$\wedge$", "$\vee$" and "$\neg$" as well as the quantifiers "$\exists$" and "$\forall$". An $\exists$-Presburger predicate is a Presburger predicate where all quantifiers are $\exists$-quantifiers that are not negated (e.g. the predicate "$\neg\exists x : x = 3$" is not a $\exists$-Presburger predicate). A quantifier-free Presburger formula has no quantifiers.

Multisets. A *multiset* over a finite set E is a mapping $M : E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M, and for every $E' \subseteq E$ we define $M(E') \stackrel{\text{def}}{=} \sum_{e \in E'} M(e)$. The empty multiset is denoted as $\mathbf{0}$. The *support* of M is defined as $[\![M]\!] \stackrel{\text{def}}{=} \{e \in E \mid M(e) > 0\}$ and *size* of M is defined as $\|M\| \stackrel{\text{def}}{=} \sum_{e \in E} M(e)$. Addition and comparison are extended to multisets componentwise, i.e. $(M + M')(e) \stackrel{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $(M \leq M')(e) \stackrel{\text{def}}{\iff} M(e) \leq M'(e)$ for every $e \in E$. We define *multiset difference* as $(M \ominus M')(e) \stackrel{\text{def}}{=} \max\{M(e) - M'(e), 0)\}$ for every $e \in E$. We sometimes denote multisets using a set-like notation, e.g. $\wr f, g, g, h \wr$ is the multiset M such that $M(f) = 1$, $M(g) = 2$, $M(h) = 1$ and $M(e) = 0$ for every $e \in E \setminus \{f, g, h\}$.

Population protocols. A (k-way) *population protocol* is a tuple $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ such that

- Q is a finite set of *states*,
- $\mathcal{T} \subseteq \bigcup_{2 \leq i \leq k} Q^i \times Q^i$ is a set of *transitions*,
- Σ is a non-empty finite input *alphabet*,
- $\mathcal{I} : \Sigma \to Q$ is the *input function* mapping input symbols to states and
- $\mathcal{O} : Q \to \{0, 1\}$ is the *output function* mapping states to boolean values.

A *configuration* of $\mathcal{P}$ is a multiset $C \in \mathbb{N}^Q$ with $C > \mathbf{0}$. For every $q \in Q$, $C(q)$ describes the number of agents in states q. A state q is *empty* if $C(q) = 0$, and otherwise *populated*.

The notation $(p_1, p_2, ..., p_i) \to (q_1, q_2, ..., q_i)$ is an alternative notation for a transition $t = ((p_1, p_2, ..., p_i), (q_1, q_2, ..., q_i))$. Intuitively, t describes that i agents in states $p_1, ..., p_i$ may interact and move to states $q_1, ..., q_i$. We will assume that every group of agents can always interact, i.e. that for all $p_1, \cdots, p_i \in Q$ there exists $q_1, \cdots, q_i \in Q$ such that $(p_1, \cdots, p_i) \to (q_1, \cdots, q_i)$. The *preset* and *postset* of t are respectively ${}^\bullet t \stackrel{\text{def}}{=} \{p_1, \cdots, p_i\}$ and $t^\bullet \stackrel{\text{def}}{=} \{q_1, \cdots q_i\}$. The notation for preset and postset is extended for sets of transitions, e.g. ${}^\bullet T \stackrel{\text{def}}{=} \bigcup_{t \in T} {}^\bullet t$. The *pre-multiset* and *post-multiset* are defined as $\text{pre}(t) = \wr p_1, \cdots, p_i \wr$ and $\text{post}(t) = \wr q_1, \cdots, q_i \wr$.

The transition t is *enabled* in configuration $C \in \mathbb{N}^Q$ if $C \geq \text{pre}(t)$. In that case it can *occur* and lead to a configuration $C' = (C \ominus \text{pre}(t) + \text{post}(t))$. Then we write $C \xrightarrow{t} C'$. A transition t is *silent* if $\text{pre}(t) = \text{post}(t)$. We write $C \to C'$ if $C \xrightarrow{t} C'$ for some $t \in \mathcal{T}$. We write $C \xrightarrow{t_1 t_2 ... t_j} C'$ if there exist $C_0, C_1, ..., C_j \in \mathbb{N}^Q$ and $t_1, t_2, ..., t_j \in \mathcal{T}$ such that $C = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} ... \xrightarrow{t_j} C_j = C'$. We write $C \xrightarrow{*} C'$ if $C \xrightarrow{\sigma} C'$ for some $\sigma \in T^*$. We say that C' is *reachable* from C if $C \xrightarrow{*} C'$. The support of a sequence $\sigma = t_1 t_2 ... t_n \in T^*$ is $[\![\sigma]\!] \stackrel{\text{def}}{=} \{t_i \mid 1 \leq i \leq n\}$. The *effect* of a transition t is a vector $\Delta_t : Q \to \mathbb{Z}$ such that $\Delta_t(q) = -\text{pre}(t)(q) + \text{post}(t)(q)$ for each state $q \in Q$.

Computing with population protocols. Every *input* $X \in \mathbb{N}^\Sigma$ is mapped to the configuration $\mathcal{I}(X) \in \mathbb{N}^Q$ as follows:

$$\mathcal{I}(X)(q) \stackrel{\text{def}}{=} \left(\sum_{a \in \Sigma : \mathcal{I}(a) = q} X(a) \right) \qquad \text{for every } q \in Q$$

A configuration C is *initial* if $C = \mathcal{I}(X)$ for some input X.

An *execution* $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \xrightarrow{t_3} \cdots$ is an infinite sequence of configurations $C_0 C_1 \cdots$ and transitions $t_1 t_2 \cdots$ such $C_i \xrightarrow{t_{i+1}} C_{i+1}$ for all $0 \leq i$. We say that π is *fair* if for every configuration D and possible step $D \xrightarrow{t} D'$ the following holds:

$$\text{If } \{i \in \mathbb{N} \mid C_i = D\} \text{ is infinite, then } \{i \in \mathbb{N} \mid C_i = D \wedge t_{i+1} = t\} \text{ is infinite.}$$

It is not hard to show the following lemma:

Lemma 2.0.1. *Let $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \xrightarrow{t_3} \cdots$ be an execution. The following two statements hold:*

1. *π is fair if and only if for every configuration D it holds that: If $\{i \in \mathbb{N} \mid C_i = D\}$ is infinite and $D \to D'$, then $\{i \in \mathbb{N} \mid C_i = D'\}$ is infinite.*

2. *π is fair, if and only if for every configuration D it holds that: If $\{i \in \mathbb{N} \mid C_i = D\}$ is infinite and $D \xrightarrow{*} D'$, then $\{i \in \mathbb{N} \mid C_i = D'\}$ is infinite.*

Intuitively, fairness ensures that a step cannot be avoided forever if it is enabled infinitely often along π.

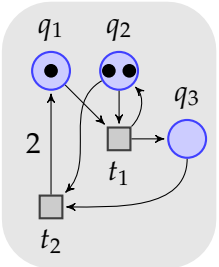
A configuration C is a *consensus configuration* if $\mathcal{O}(p) = \mathcal{O}(q)$ for all $p, q \in [\![C]\!]$. If a configuration C is a consensus configuration, then its output $\mathcal{O}(C)$ is the unique output of its states, otherwise it is $\perp$. An execution $\pi = C_0 C_1 ...$ stabilizes to $b \in \{0,1\}$ if $\mathcal{O}(C_i) = \mathcal{O}(C_{i+1}) = ... = b$ for some $i \in \mathbb{N}$. The output of π is $\mathcal{O}(\pi) \stackrel{\text{def}}{=} b$ if it stabilizes to b, and $\mathcal{O}(\pi) \stackrel{\text{def}}{=} \perp$ otherwise. A consensus configuration C is *stable* if every configuration C' reachable from C is a consensus configuration with $\mathcal{O}(C') = \mathcal{O}(C)$. It can easily be shown that a fair execution stabilizes to $b \in \{0,1\}$ if and only if it contains a stable configuration whose output is b.

A population protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ is *well-specified* if for every initial configuration C_0, there exists $b \in \{0,1\}$ such that every fair execution π starting at C_0 has output b. If $\mathcal{P}$ is well-specified, then we say that it *computes the predicate* $\varphi : \mathbb{N}^\Sigma \to \{0,1\}$ if for every input X, every fair execution of $\mathcal{P}$ starting at $\mathcal{I}(X)$ stabilizes to $\varphi(X)$.

Silent vs. non-silent. A configuration is *terminal* if it only enables silent transitions. A population protocol is *silent* if every fair execution eventually reaches a terminal configuration. From this point on, the configuration cannot change anymore. In contrast, a protocol is *non-silent* if there is a fair execution that does not reach a terminal configuration. Examples for a silent and a non-silent population protocol can be seen in Figures 2.1 and 2.2, respectively.

Graphical representation. Population protocols are closely related to Petri nets. Thus one can graphically represent a population protocol in the standard Petri net notation i.e. states are circles and transitions are squares. A configuration of a population protocol corresponds to a marking of the drawn Petri net.

$$
\begin{aligned}
Q &= \{q_1, q_2, q_3\} \\
\mathcal{T} &= \{t_1 : q_1, q_2 \to q_2, q_3, \\
&\qquad t_2 : q_2, q_3 \to q_1, q_1\} \\
C &= \wr q_1, q_2, q_2 \int
\end{aligned}
\qquad \longleftrightarrow
$$



$$
\begin{aligned}
Q &= \{A, B, a, b\} \\
\mathcal{T} &= \{t_{AB} : A, B \to a, b \\
&\quad\ t_{Ab} : A, b \to A, a \\
&\quad\ t_{Ba} : B, a \to B, b \\
&\quad\ t_{Ab} : a, b \to b, b\} \\
\Sigma &= \{A, b\} \\
\mathcal{I}(A) &= A \qquad \mathcal{I}(B) = B \\
\mathcal{O}(A) &= \mathcal{O}(a) = 0 \\
\mathcal{O}(B) &= \mathcal{O}(b) = 1
\end{aligned}
$$

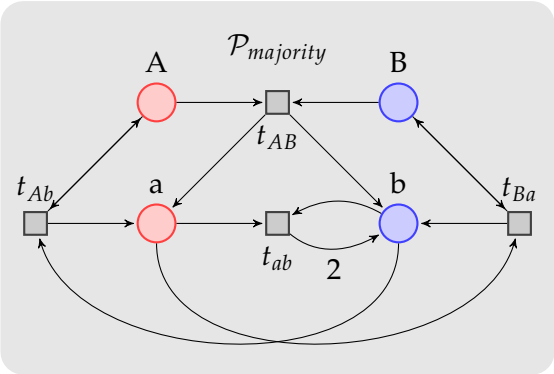


Figure 2.1.: Silent population protocol $\mathcal{P}_{majority}$ for "$A \leq B$".

$$
\begin{aligned}
Q &= \{q_0, q_1, q_2, q_3\} \\
\mathcal{T} &= \{t_{11} : q_1, q_1 \to q_2, q_0 \\
&\quad\ t_{02} : q_0, q_2 \to q_1, q_1 \\
&\quad\ t_{12} : q_1, q_2 \to q_3, q_3 \\
&\quad\ t_{03} : q_0, q_3 \to q_3, q_3 \\
&\quad\ t_{13} : q_1, q_3 \to q_3, q_3 \\
&\quad\ t_{23} : q_2, q_3 \to q_3, q_3\} \\
\Sigma &= \{X\} \\
\mathcal{I}(X) &= q_1 \\
\mathcal{O}(q_3) &= 1 \\
\mathcal{O}(q_0) &= \mathcal{O}(q_1) = \mathcal{O}(q_2) = 0
\end{aligned}
$$

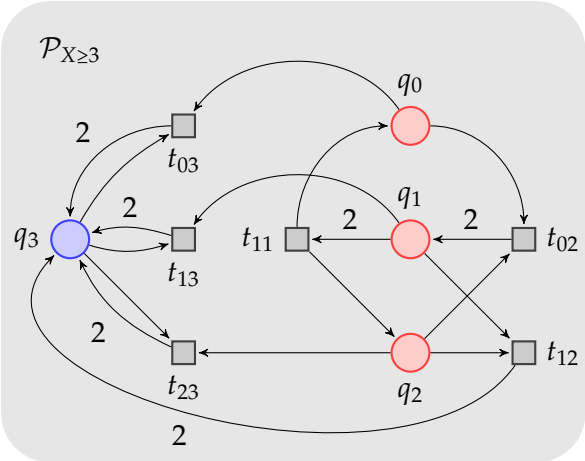


Figure 2.2.: Non-silent population protocol $\mathcal{P}_{X\geq 3}$ for "$X \geq 3$".

Examples. We will use two population protocols as running examples in this work.

One of the protocols is the majority protocol $\mathcal{P}_{majority}$ of Figure 2.1. This silent population protocol answers the question whether there are initially at least as many agents in state B as in state A. First, it cancels out the agents of states A and B in pairs.

If one side wins, the remaining agents can change the output of all other agents. The transition t_{ab} produces the desired output if there is a tie.

The second protocol is the Flock-of-Birds protocol $\mathcal{P}_{X\geq 3}$ of Figure 2.2 that computes the predicate $X \geq 3$. If there are at least 3 agents, then fairness ensures that the transition t_{12} is fired. Afterwards all agents will be moved to state q_3 via the transitions t_{03}, t_{13} and t_{23}. If there are less than 3 agents, then the transition t_{12} cannot be fired and all agents stay in the states with output *false*. Note, that in this case, it is possible to never reach a terminal configuration. Thus, the protocol is non-silent. The protocol is an instance of the protocol family *reversible succinct Flock-of-Birds* that is described in Section A.2.

Linear temporal logic in population protocols. To specify properties of fair executions in population protocols, we introduce a version of *linear temporal logic* (LTL). Let τ be a Presburger predicate with the free variables q for each state $q \in Q$ and $\varphi, \varphi_1, \varphi_2$ be LTL predicates. The syntax of an LTL formula φ is defined by the grammar:

$$\varphi := \tau \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid F\varphi \mid G\varphi$$

Furthermore, we allow the abbreviation

$$\varphi_1 \Rightarrow \varphi_2 \stackrel{\text{def}}{=} \neg\varphi_1 \vee \varphi_2.$$

For a configuration C of the protocol $\mathcal{P}$ the semantics of LTL predicates are defined as follows:

$$\begin{aligned}
C \vDash \tau &\quad\Leftrightarrow\quad \tau(C)\\
C \vDash \neg\varphi &\quad\Leftrightarrow\quad C \nvDash \varphi\\
C \vDash (\varphi_1 \wedge \varphi_2) &\quad\Leftrightarrow\quad (C \vDash \varphi_1) \wedge (C \vDash \varphi_2)\\
C \vDash (\varphi_1 \vee \varphi_2) &\quad\Leftrightarrow\quad (C \vDash \varphi_1) \vee (C \vDash \varphi_2)\\
C \vDash X\varphi &\quad\Leftrightarrow\quad \forall C' : C \to C' \Rightarrow C' \vDash \varphi\\
C \vDash F\varphi &\quad\Leftrightarrow\quad \forall \text{ fair } \pi = C_0C_1\cdots \text{ with } C_0 = C : \exists k \geq 0 : C_k \vDash \varphi\\
C \vDash G\varphi &\quad\Leftrightarrow\quad \forall \text{ fair } \pi = C_0C_1\cdots \text{ with } C_0 = C : \forall k \geq 0 : C_k \vDash \varphi
\end{aligned}$$

We extend the notation for a set of configurations $\mathscr{C}$:

$$\mathscr{C} \vDash \varphi \Leftrightarrow (\forall C \in \mathscr{C} : C \vDash \varphi)$$

Intuitively, an LTL formula holds for a set of configurations if and only if it holds for each configuration individually.

3. Stage Graphs

Stage graphs are a representation of the hierarchical structure of population protocols. A stage graph describes all possible fair executions of a protocol by grouping configurations into finitely many *stages*. These stages correspond to non-reversible changes in the configuration of agents. The concept of stage graphs was first introduced by Blodin *et al.* in [8]. In their work, stage graphs where used to automatically bound the speed of a protocol. In addition to this application (see Section 5), we use stage graphs to automatically verify properties of population protocols like correctness and even liveness (see Section 6).

A set of configurations $\mathscr{C}$ is *inductive* if $C \in \mathscr{C}$ and $C \to C'$ implies $C' \in \mathscr{C}$. An execution $\pi = C_0C_1\cdots$ *enters* a set of configurations $\mathscr{C}$ if $C_i \in \mathscr{C}$ for some $i \geq 0$.

Definition 1 (Stage graph)**.** *Let $\mathcal{P}$ be a population protocol and let φ_{pre} and φ_{post} be Presburger predicates. A* $(\varphi_{pre}, \varphi_{post})$-stage-graph *for $\mathcal{P}$ is a directed acyclic graph whose vertices are sets of configurations of $\mathcal{P}$, called* stages*. The successors of a stage $\mathscr{C}$ are called* substages *of $\mathscr{C}$. If a stage has no substages, it is* terminal*, otherwise it is* non-terminal*. In a $(\varphi_{pre}, \varphi_{post})$-stage-graph the following conditions are satisfied:*

1. *Every stage is an inductive set of configurations.*
2. *Every configuration $C \vDash \varphi_{pre}$ belongs to some stage.*
3. *For every non-terminal stage $\mathscr{C}$ with substages $\mathscr{C}_1, \ldots, \mathscr{C}_n$ and configuration $C \in \mathscr{C}$, there exists a configuration $C' \in \mathscr{C}_1 \cup \cdots \cup \mathscr{C}_n$ such that $C \xrightarrow{*} C'$.*
4. *For every terminal stage $\mathscr{C}$ it holds that $\mathscr{C} \vDash \varphi_{post}$.*

This definition implies that if a stage $\mathscr{C}_{sub}$ is a substage of some stage $\mathscr{C}$, then $\mathscr{C}_{sub} \subseteq \mathscr{C}$.

Theorem 3.0.1. *A population protocol $\mathcal{P}$ has a $(\varphi_{pre}, \varphi_{post})$-stage-graph if and only if $\mathbb{N}^Q \vDash \varphi_{pre} \Rightarrow FG(\varphi_{post})$.*

Proof. "⇒": Assume $\mathcal{P}$ has a $(\varphi_{pre}, \varphi_{post})$-stage-graph. Let $C_0 \in \mathbb{N}^Q$ be some configuration. If $C_0 \nvDash \varphi_{pre}$, the LTL formula is satisfied. Therefore, assume $C_0 \vDash \varphi_{pre}$. Let $\pi = C_0C_1\cdots$ be some fair execution of $\mathcal{P}$. Using condition 2, we know that C_0 is part of some stage $\mathscr{C}$.

We will now argue that π eventually enters a terminal stage. With condition 1 we know that π remains in $\mathscr{C}$. For a given input of size n there are only finitely many distinct configurations possible. As π is infinite, we know that at least one configuration $C \in \mathscr{C}$ is visited infinitely often. There exists a configuration C' with $C \xrightarrow{*} C'$ that is part of a substage (condition 3). Because of fairness and Lemma 2.0.1 π must visit C' and therefore enter a substage. As there are only finitely many stages in a stage graph and it is acyclic, we know that π eventually enters a terminal stage $\mathscr{C}_{term}$.

From that moment on it remains in $\mathscr{C}_{term}$ (condition 1). Because of condition 4 all following configurations satisfy φ_{post}.

"$\Leftarrow$": Assume $\mathbb{N}^Q \vDash \varphi_{pre} \to FG(\varphi_{post})$. There is a $(\varphi_{pre}, \varphi_{post})$-stage-graph with two stages C_{init} and its substage C_{term}. $C_{init} \stackrel{\text{def}}{=} \left\{ C \,\middle|\, \exists C_0 \vDash \varphi_{pre} \wedge C_0 \xrightarrow{*} C \right\}$ is the set of configurations reachable from some configuration that satisfies φ_{pre}. $C_{term} \stackrel{\text{def}}{=} \left\{ C \,\middle|\, \forall C' \text{ with } C \xrightarrow{*} C' : C' \vDash \varphi_{post} \right\}$ is the set of configurations that can only reach configurations that satisfy φ_{post}. Both stages are inductive by definition (condition 1). $\mathscr{C}_{init}$ contains every configuration $C \vDash \varphi_{pre}$ (condition 2). The only terminal stage $\mathscr{C}_{term}$ satisfies condition 4 by definition.

We still need to show condition 3. Assume for sake of contradiction that there is a $C \in \mathscr{C}_{init}$ that cannot reach $\mathscr{C}_{term}$. By the definition of $\mathscr{C}_{init}$, there exists a $C_0 \vDash \varphi_{pre}$ and finite sequence σ such that $C_0 \xrightarrow{\sigma} C$. σ cannot visit $\mathscr{C}_{term}$ because then C would be in $\mathscr{C}_{term}$ using condition 1. Now we choose any fair execution π' starting at C. By assumption π' does not visit $\mathscr{C}_{term}$. We construct a new execution $\pi = \sigma\pi'$. π is fair because π' is fair and π and π' visit the same set of configurations infinitely often. The execution π is fair, starts at $C_0 \vDash \varphi_{pre}$ but does not visit $\mathscr{C}_{term}$. This contradicts the initial assumption. We conclude that condition 3 is satisfied. □

The following theorem shows that stage graphs can serve as a certificate for proving correctness of a population protocol.

Theorem 3.0.2. *Let Λ be a predicate. For $b \in \{0,1\}$ let*

$$\varphi_{init,b}(C) \stackrel{def}{=} \exists X \in \mathbb{N}^\Sigma : (\Lambda(X) = b) \wedge (\mathcal{I}(X) = C)$$
$$\varphi_{out,b}(C) \stackrel{def}{=} (\mathcal{O}(C) = b).$$

A population protocol $\mathcal{P}$ has a $(\varphi_{init,0}, \varphi_{out,0})$-stage-graph and a $(\varphi_{init,1}, \varphi_{out,1})$-stage-graph if and only if it computes the predicate Λ.

Proof. "$\Rightarrow$": Assume there is a $(\varphi_{init,0}, \varphi_{out,0})$-stage-graph and a $(\varphi_{init,1}, \varphi_{out,1})$-stage-

graph. Using Theorem 3.0.1, we know:

$$\mathbb{N}^Q \vDash \varphi_{init,0} \to FG(\varphi_{out,0})$$
$$\mathbb{N}^Q \vDash \varphi_{init,1} \to FG(\varphi_{out,1})$$

Using the definition of $\varphi_{init,0}$ and $\varphi_{init,1}$, we conclude that $\mathcal{P}$ is well-specified and converges to the correct output.

"$\Leftarrow$": If $\mathcal{P}$ computes Λ we know that any fair execution starting in some initial configuration $C_0 \vDash \varphi_{init,b}$ stabilizes to the lasting consensus b. For any configuration C with lasting consensus b we know $C \vDash \varphi_{out,b}$. By applying Theorem 3.0.1, we get the desired result. □

3.1. Example

Recall the example protocol $\mathcal{P}_{majority}$ of Figure 2.1. The stage graph in Figure 3.1 verifies that every execution of $\mathcal{P}_{majority}$ always results in a lasting consensus. S_4 corresponds to a majority of agents in state B. S_9 corresponds to a majority of agents in state A. The stage S_8 represents the cases where there is a tie. This exact stage graph can be automatically computed using the algorithm that is described in Section 4. The result can be seen in Figure 4.5.

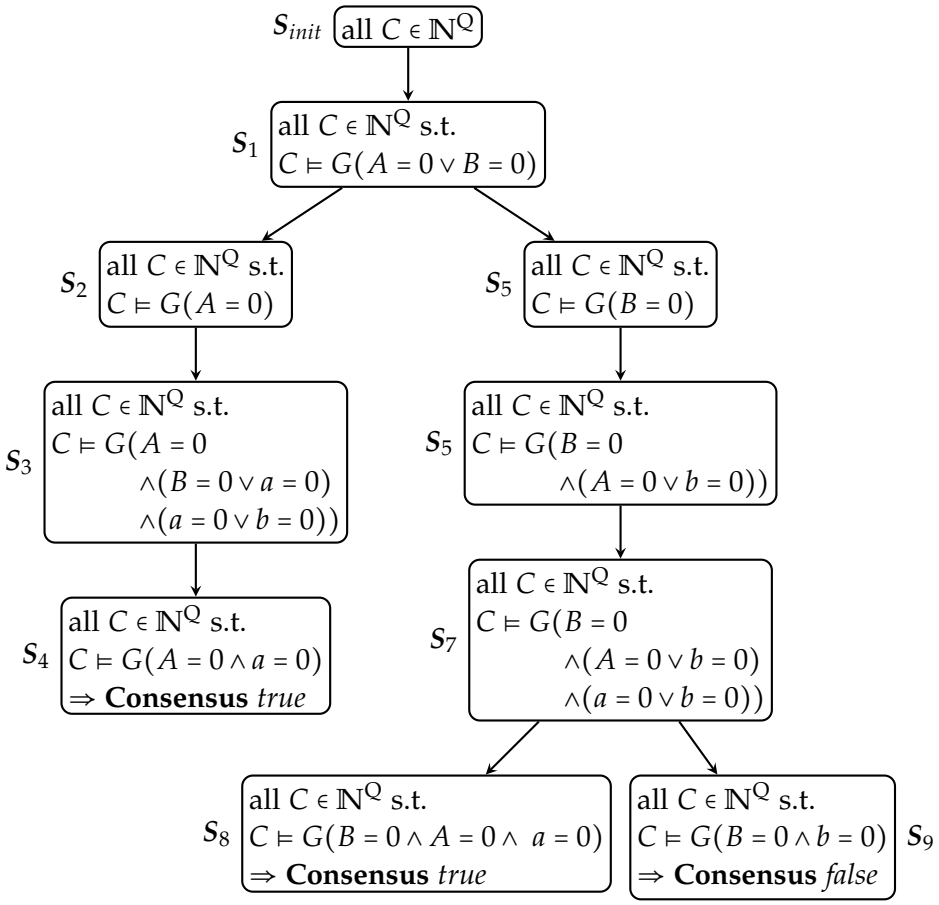


Figure 3.1.: Stage graph for consensus of $\mathcal{P}_{majority}$.

4. Computing Stage Graphs

Deciding whether a protocol computes a given predicate has very high complexity (at least *TOWER*-hard [3][4]). Therefore, we design an algorithm that might fail but has lower complexity and works for many protocols in the literature.

The main idea is to automatically compute a stage graph that verifies the desired property. The stages of the generated graph are defined using the concept of *deserted states* and *dead transitions*. Intuitively, a transition is dead if it cannot be enabled again and a state is deserted if it cannot be populated again.

Definition 2. *Let C be a configuration.*

- *A state q is* deserted *at C if $C \vDash G(q = 0)$. The set of states deserted at C is denoted* $\textsc{Deserted}(C)$. *Given a set of configurations $\mathscr{C}$, let* $\textsc{Deserted}(\mathscr{C}) = \bigcap_{C \in \mathscr{C}} \textsc{Deserted}(C)$.
- *A transition t is* dead *at C if $C \vDash G\left(\bigvee_{q \in \bullet t(q)} q < \mathrm{pre}(t)(q)\right)$. The set of transitions dead at C is denoted* $\textsc{Dead}(C)$. *Given a set of configurations $\mathscr{C}$, let* $\textsc{Dead}(\mathscr{C}) = \bigcap_{C \in \mathscr{C}} \textsc{Dead}(C)$.

Figure 4.1 shows an example for a configuration with deserted states and dead transitions. State q is empty and has no incoming transitions. Therefore, it is deserted and t must be dead. Transition u shows that a transition can be dead although no state in the preset is deserted.

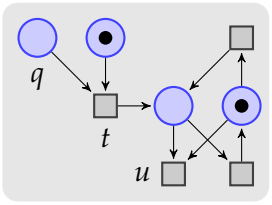


Figure 4.1.: Example configuration with deserted state q and dead transitions t and u.

We define some LTL constraints. Let t be a transition, $\mathcal{T}$ be a set of transitions, q be a

state and Q be a set of states.

$$\textsc{Enabled}(t) \stackrel{\text{def}}{=} \bigwedge_{q \in {}^\bullet t} q \geq \text{pre}(t)(q)$$

$$\textsc{Dis}(t) \stackrel{\text{def}}{=} \neg\textsc{Enabled}(t)$$

$$\textsc{Dis}(\mathcal{T}) \stackrel{\text{def}}{=} \bigwedge_{t \in \mathcal{T}} \textsc{Dis}(t)$$

$$\textsc{Dead}(t) \stackrel{\text{def}}{=} G(\textsc{Dis}(t))$$

$$\textsc{Dead}(\mathcal{T}) \stackrel{\text{def}}{=} \bigwedge_{t \in \mathcal{T}} \textsc{Dead}(t)$$

$$\textsc{Empty}(q) \stackrel{\text{def}}{=} q = 0$$

$$\textsc{Empty}(\mathrm{Q}) \stackrel{\text{def}}{=} \bigwedge_{q \in \mathrm{Q}} \textsc{Empty}(q)$$

$$\textsc{Deserted}(q) \stackrel{\text{def}}{=} G(\textsc{Empty}(q))$$

$$\textsc{Deserted}(\mathrm{Q}) \stackrel{\text{def}}{=} \bigwedge_{q \in \mathrm{Q}} \textsc{Deserted}(q)$$

The following algorithm is inspired by the way most protocols in the literature are designed. Protocols work in phases. Initially, all transitions of the protocol can occur. The end of a phase is marked by the "death" of one or more transitions. This could result in some states becoming deserted. After several phases we reach a phase where the postcondition holds.

4.1. Stage Representation

We use formulas to represent possibly infinite sets of configurations. All formulas produced by this algorithm are in fact Presburger formulas. A stage of a $(\varphi_{pre}, \varphi_{post})$-stage-graph for the protocol $\mathcal{P} = (\mathrm{Q}, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ is a tuple $S = (T_{dead}, Q_{deserted})$. $T_{dead} \subseteq \mathcal{T}$ is the set of dead transitions in S and $Q_{deserted} \subseteq \mathrm{Q}$ is the set of deserted states in S.

A configuration C is in stage S if $(\exists C_0 \vDash \varphi_{pre} : C_0 \xrightarrow{*} C) \wedge T_{dead} \subseteq \textsc{Dead}(C) \wedge Q_{deserted} \subseteq \textsc{Deserted}(C)$. In that case we write $C \in S$.

4.2. Building Blocks

The algorithm generating a $(\varphi_{pre}, \varphi_{post})$-stage-graph for a protocol $\mathcal{P}$ is parametric in three auxiliary functions. Let $S = (T_{dead}, Q_{deserted})$ be a stage.

- EventuallyDead$(\mathcal{P}, \varphi_{pre}, S)$: Returns a set of transitions $T'_{dead} (\supseteq T_{dead})$ that eventually become dead in any fair execution starting in some configuration C of the stage S:
  $S \vDash F(\text{Dead}(T'_{dead}))$
- Terminal$(\mathcal{P}, \varphi_{pre}, S, \varphi_{post})$: Checks whether the stage S is terminal. It must hold that Terminal$(\mathcal{P}, \varphi_{pre}, S, \varphi_{post}) \Rightarrow S \vDash \varphi_{post}$. (Note that this is only an implication. Therefore, it may return *false* even if the stage is terminal.)
- Split$(\mathcal{P}, \varphi_{pre}, S)$: Either fails or returns a set of substages $S_1, \cdots, S_k$ where $S_i = (T_{dead_i}, Q_{deserted_i})$ such that each substage has strictly more deserted states than S. Additionally, every configuration of S needs to be part of some substage:
  $(\bigwedge_{i=1}^{k} Q_{deserted} \subset Q_{deserted_i}) \wedge (S \subseteq \bigcup_{i=1}^{k} S_i)$

4.3. The Algorithm

The main algorithm (Listing 4.1) starts with the initial stage $S_0 = (\varnothing, \varnothing)$. As long as there are unprocessed stages, it will compute their substages using the main procedure Substages of Listing 4.2. If one of the calls fails, then the whole algorithm fails.

Listing 4.1: "Algorithm for computing a φ_{pre}-φ_{post}-stage-graph"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       Presburger predicate φ_post

S_0 := (∅, ∅)
Unprocessed := {S_0}

while |Unprocessed| > 0
    S := Unprocessed.pop()

    if Substages(P, φ_pre, φ_post, S) fails
        then abort
    else
        Unprocessed := Unprocessed ∪ Substages(P, φ_pre, φ_post, S)
```

Listing 4.2: "Procedure: SUBSTAGES"

```
input: protocol 𝒫 = (Q, 𝒯, Σ, ℐ, 𝒪)
       Presburger predicate φ_pre
       Presburger predicate φ_post
       stage S = (T_dead, Q_deserted)

if TERMINAL(𝒫, φ_pre, S, φ_post)
    return ∅

T'_dead := EVENTUALLYDEAD(𝒫, φ_pre, S)
if T'_dead ⊃ T_dead
    return {(T'_dead, Q_deserted)}

if SPLIT(𝒫, φ_pre, S) fails
    then abort
else
    return SPLIT(𝒫, φ_pre, S)
```

Theorem 4.3.1. *A successful run of the algorithm in Listing 4.1 constructs a* $(\varphi_{pre}, \varphi_{post})$*-stage-graph.*

Proof. We have to prove conditions 1 to 4 of the definition of stage graph.

1. Every constructed stage is inductive by definition of the stage representation.
2. Every configuration $C \vDash \varphi_{pre}$ is in the initial stage by definition of the stage representation.
3. Let $S = (T_{dead}, Q_{deserted})$ be a non-terminal stage with substages $S_1, \cdots, S_k$.

   In the case that EVENTUALLYDEAD returned a set $T'_{dead} (\supseteq T_{dead})$, there is exactly one substage $S_1 = (T'_{dead}, Q_{deserted})$. By the definition of EVENTUALLYDEAD, we know: $S \vDash F(\text{DEAD}(T'_{dead}))$. Therefore, for every C there exists a $C' \vDash \text{DEAD}(T'_{dead})$ such that $C \xrightarrow{*} C'$. As $C' \in S_1$, condition 3 holds.

   If the substages where the result of a successful SPLIT call, we know that every $C \in S$ is in $S_1 \cup \cdots \cup S_k$. Therefore, the condition is satisfied by setting $C' = C$.
4. In a successful run of the algorithm, a stage S can only have no substage if TERMINAL was true. By the definition of TERMINAL, the condition is satisfied.

□

4.4. Implementation of Building Blocks

In the following section, we give an implementation of the functions called by the algorithm. Our implementation makes use of an SMT solver.

Largest empty siphon. Multiple of the following Presburger constraints use the notion of a largest empty siphon (or trap).

Definition 3 (Traps & Siphons). *Let $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ be a population protocol.*

- *A subset of states $P \in Q$ is a* trap *if $P^\bullet \subseteq {}^\bullet P$.*
- *A subset of states $P \in Q$ is a* siphon *if ${}^\bullet P \subseteq P^\bullet$.*

Intuitively, non-empty traps stay non-empty because all transitions that consume agents from the trap also add agents to the trap. Empty siphons stay empty because all transitions that add agents to the siphon also need to consume agents from the siphon.

We need a constraint LargestEmptySiphon$(\mathcal{P}, C, P)$ for a population protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$, a configuration C and a set of states $P \subseteq Q$, that is satisfiable if and only if P is the largest empty siphon in configuration C.

Listing 4.3: "Procedure: LargestEmptySiphon"

```
input: protocol P = (Q, T, Σ, I, O)
       configuration C

n := |Q|

# remove non-empty states (step 0)
X := Q ∖ ⟦C⟧

for i from 1 to n
    # remove states that violate siphon condition (step i)
    R := {q ∈ X | ∃t ∈ T : q ∈ t• ∧ •t ∩ X = ∅}
    X := X ∖ R

return X
```

The basis for the constraint is the algorithm in Listing 4.3 that computes the largest siphon [9] [10]. We symbolically execute the algorithm by encoding its execution as a constraint. Let r be a vector of unknowns containing one unknown $r(q)$ for each state $q \in Q$ and let $n = |Q|$. For a protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$, a configuration C and a

set of states $P \subseteq Q$ we define $\textsc{LargestEmptySiphon}(\mathcal{P}, C, P)$ as the following system of constraints:

$$C(q) = 0 \qquad \text{for all } q \in P \tag{4.1}$$

$$t^\bullet \cap P \neq \varnothing \Rightarrow {}^\bullet t \cap P \neq \varnothing \qquad \text{for all } t \in \mathcal{T} \tag{4.2}$$

$$r(q) = n \qquad \text{for all } q \in P \tag{4.3}$$

$$0 \leq r(q) < n \qquad \text{for all } q \in Q \setminus P \tag{4.4}$$

$$r(q) = 0 \qquad \text{for all } q \in [\![C]\!] \tag{4.5}$$

$$0 < r(q) < n \Rightarrow \bigvee_{t \in \mathcal{T}} \left(q \in t^\bullet \wedge \bigwedge_{q' \in {}^\bullet t} r(q') < r(q) \right) \qquad \text{for all } q \in P \tag{4.6}$$

Intuitively, we guess $r(q)$ to be the iteration where q is removed from the siphon. $r(q) = 0$ tells us that q was removed in iteration 0 because it was not empty. $r(q) = n$ tells us that q was never removed and must therefore be part of the siphon. Otherwise, there must be a transition t that led to the removal of q.

Note that we can swap the pre- and postsets in the algorithm of Listing 4.3 to calculate the largest empty trap for a given protocol $\mathcal{P}$ and a configuration C. Thus, we define the constraint $\textsc{LargestEmptyTrap}(\mathcal{P}, C, P)$ by swapping pre- and postsets of the constraint $\textsc{LargestEmptySiphon}(\mathcal{P}, C, P)$. $\textsc{LargestEmptyTrap}(\mathcal{P}, C, P)$ is satisfiable if and only if P is the largest empty trap in configuration C.

Overapproximation of stages. Let $\mathcal{P}$ be a population protocol and let φ_{pre} be a (Presburger) precondition. Our implementation relies on a constraint $\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S)$ that holds for any configuration of stage $S = (T_{dead}, Q_{deserted})$:

$$C \in S \quad \Rightarrow \quad C \vDash \textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S)$$

This is in fact an overapproximation of the stage.

Recall that $C \in S \Leftrightarrow (\exists C_0 \vDash \varphi_{pre} : C_0 \xrightarrow{*} C) \wedge T_{dead} \subseteq \textsc{Dead}(C) \wedge Q_{deserted} \subseteq \textsc{Deserted}(C)$. One of the reasons for the overapproximation is the high complexity of the reachability problem in population protocols. The other reason is that there is no trivial constraint that guarantees that transitions are dead. To determine if a transition is dead, one needs to make sure that following steps do not reenable the transition.

We define:

$$\begin{aligned} \textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \overset{\text{def}}{=} \quad & \exists C_0 \vDash \varphi_{pre} : \textsc{PotReach}(\mathcal{P}, C_0, C) \\ & \wedge \textsc{PotDead}(\mathcal{P}, T_{dead}) \\ & \wedge \textsc{PotDeserted}(\mathcal{P}, Q_{deserted}) \end{aligned}$$

We use the notion of *potential reachability* from [5]. Blondin *et al.* combine the flow equation with siphon and trap constraints to define a constraint $\textsc{PotReach}(\mathcal{P}, C_0, C)$ of integer linear arithmetic such that:

$$(C_0 \xrightarrow{*} C) \Rightarrow \textsc{PotReach}(\mathcal{P}, C_0, C).$$

For the protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ we define:

$$\begin{aligned}
\textsc{PotReach}(\mathcal{P}, C_0, C) \stackrel{\text{def}}{=} \exists x \in \mathbb{N}^{\mathcal{T}}, Q_{siphon} \subseteq Q, Q_{trap} \subseteq Q : & && (4.7)\\
& \bigwedge_{q \in Q} C(q) = C_0(q) + \sum_{t \in \mathcal{T}} \Delta_t(q) \cdot x(t) && (4.8)\\
& \wedge \textsc{LargestEmptySiphon}((Q, [\![x]\!], \Sigma, \mathcal{I}, \mathcal{O}), C_0, Q_{siphon}) && (4.9)\\
& \wedge \bigwedge_{q \in Q_{siphon}} C(q) = 0 && (4.10)\\
& \wedge \bigwedge_{t \in \mathcal{T}} {}^{\bullet}t \cap Q_{siphon} \Rightarrow x(t) = 0 && (4.11)\\
& \wedge \textsc{LargestEmptyTrap}((Q, [\![x]\!], \Sigma, \mathcal{I}, \mathcal{O}), C, Q_{trap}) && (4.12)\\
& \wedge \bigwedge_{q \in Q_{trap}} C_0(q) = 0 && (4.13)\\
& \wedge \bigwedge_{t \in \mathcal{T}} t^{\bullet} \cap Q_{trap} \Rightarrow x(t) = 0 && (4.14)
\end{aligned}$$

Intuitively, we guess $x(t)$ to be the number of times the transition t is used. We guess Q_{siphon} to be the largest empty siphon in the initial configuration and Q_{trap} to be the largest empty trap in the final configuration. The flow equation guarantees that the overall effect of the transitions matches the difference between final and initial configuration (4.8). The constraints (4.9), (4.10), (4.11) exclude states and transitions corresponding to initially unmarked siphons. The constraints (4.12), (4.13), (4.14) make sure that all traps that need to be empty in the final configuration are never filled.

To overapproximate the configurations where some set of states $Q_{deserted}$ of protocol $\mathcal{P}$ is deserted, we define:

$$\textsc{PotDeserted}(\mathcal{P}, Q_{deserted}) \stackrel{\text{def}}{=} \textsc{Empty}(Q_{deserted})$$

This is an overapproximation as a state might be empty but can be filled again.

There is a similar overapproximation for the configurations where a set of transitions T_{dead} is dead.

$$\textsc{PotDead}(\mathcal{P}, T_{dead}) \stackrel{\text{def}}{=} \textsc{Dis}(T_{dead})$$

We also introduce a more precise implementation $\textsc{PotDead}_{backwards}(\mathcal{P}, T_{dead})$ that uses more computational resources. The idea is to compute the set of configurations where

the transitions T_{dead} are dead using the backwards coverability algorithm [11][12] (see Listing 4.4).

Listing 4.4: "Procedure: BACKWARDS"

```
input: protocol P = (Q, T, Σ, I, O)
       transitions T_dead

result := {}
WS := {pre(t) | t ∈ T_dead}        # enable some dead transition

while WS ≠ ∅
    result := result ∪ WS

    # only keep minimal configurations (--> upward-closed set)
    while ∃m, m' ∈ results : m' ≤ m
        result := result ∖ {m}

    # find configurations that can cover some minimal
      configuration
    WS := {}
    for each m ∈ results
        for each t ∈ T ∖ T_dead
            m_new := (m ⊖ post(t)) + pre(t)      # i.e. m_new --t--> m
            if ∀m' ∈ result : m' ≰ m_new
                # not yet in upward-closed set
                WS := WS ∪ {m_new}

return result
```

Let C, C' be configurations such that $C \leq C'$. Assume that there is a sequence $\sigma \in \mathcal{T}^*$ such that $C \xrightarrow{\sigma} C''$ and $C'' \notin \text{Dis}(T_{dead})$. In that case, it must hold that $C' \xrightarrow{\sigma} C'' + (C' \ominus C) \geq C''$. Therefore, $C'' + (C' \ominus C) \notin \text{Dis}(T_{dead})$. Thus, the set of configurations where T_{dead} is not dead is upward-closed.

The algorithm in Listing 4.4 maintains an upward-closed set (represented by a set of minimal configurations) that can enable some transition that is supposed to be dead. Then, it iteratively computes configurations that can cover one of those minimal configurations and adds them to the upward-closed set. The resulting upward-closed set precisely contains all configurations where some $t \in T_{dead}$ is not yet dead. This allows us to compute a constraint that precisely describes the configurations where

T_{dead} are dead:

$$\textsc{PotDead}_{backwards}(\mathcal{P}, T_{dead}) \stackrel{\text{def}}{=} \bigwedge_{m \in \textsc{Backwards}(\mathcal{P}, T_{dead})} \bigvee_{q \in [\![m]\!]} q < m(q)$$

For a stage $S = (T_{dead}, Q_{deserted})$ the two resulting alternative implementations of $\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S)$ are:

$$\begin{aligned}
\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \stackrel{\text{def}}{=} \quad & \exists C_0 \vDash \varphi_{pre} : \textsc{PotReach}(\mathcal{P}, C_0, C) \\
& \wedge \textsc{PotDead}(\mathcal{P}, T_{dead}) \\
& \wedge \textsc{PotDeserted}(\mathcal{P}, Q_{deserted}) \\
\textsc{PotInStage}_{backwards}(\mathcal{P}, \varphi_{pre}, S) \stackrel{\text{def}}{=} \quad & \exists C_0 \vDash \varphi_{pre} : \textsc{PotReach}(\mathcal{P}, C_0, C) \\
& \wedge \textsc{PotDead}_{backwards}(\mathcal{P}, T_{dead}) \\
& \wedge \textsc{PotDeserted}(\mathcal{P}, Q_{deserted}).
\end{aligned}$$

The constraint $\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S)$ has size $POLY(|\mathcal{P}| + |\varphi_{pre}|)$. If the precondition φ_{pre} is a $\exists$-Presburger predicate, then the whole constraint is a $\exists$-Presburger predicate and deciding $C \vDash \textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S)$ is in NP. The same holds for the constraint $\textsc{PotInStage}_{backwards}(\mathcal{P}, \varphi_{pre}, S)$, however, its size can be as large as double exponential in $|\mathcal{P}| + |\varphi_{pre}|$ because of the backwards coverability algorithm [13].

4.4.1. EventuallyDead

We give three different implementations of this function. The first one uses *ranking functions* to find transitions that are eventually dead. The second one uses the concept of *layered termination*. The third implementation combines both approaches.

Ranking Function

We try to find a ranking function that shows that a set of transitions can only fire a finite number of times.

Lemma 4.4.1. *Let $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ be a population protocol. Let $S = (T_{dead}, Q_{deserted})$ be a stage. Let $E = \mathcal{T} \setminus T_{dead}$ and $D \subseteq E$ be a set of transitions. Let y be a vector of unknowns containing one unknown $y(q)$ for every state $q \in Q$. If the system of inequations*

$$y \geq 0 \tag{4.15}$$

$$\sum_{q \in Q} y(q) \cdot \Delta_t(q) < 0 \qquad \text{for all } t \in D \tag{4.16}$$

$$\sum_{q \in Q} y(q) \cdot \Delta_t(q) \leq 0 \qquad \text{for all } t \in E \setminus D \tag{4.17}$$

has a rational solution, then every run starting at any $C \in S$ contains at most $O(|C|)$ occurrences of transitions that are in D and $C \vDash F(\textsc{Dead}(D))$.

Proof. Assume that the system of inequations has a rational solution y. Let $C_0 \in S$ and let $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \cdots$ be an infinite run starting at C_0. We necessarily have $t_i \in E$ for all $i \geq 0$.

We define the ranking function $r(C) \stackrel{\text{def}}{=} \sum_{q \in Q} y(q) \cdot C(q)$. Because of (4.16) and (4.17), firing a transition $t \in E$ does not increase the ranking and firing a transition $t \in D$ decreases the ranking. The minimal decrease in ranking when firing a transition $t \in D$ is

$$\min_{t \in D}(-\sum_{q \in Q} y(q) \cdot \Delta_t(q)).$$

As $y \geq 0$, we have $R(C_i) \geq 0$ for any $i \geq 0$. Therefore, the ranking can at most decrease by $R(C_0)$. Furthermore we know that $R(C_0) \leq |C_0| \cdot \max_{q \in Q}(y(q))$. Thus there can be at most

$$\frac{|C_0| \cdot \max_{q \in Q}(y(q))}{\min_{t \in D}(-\sum_{q \in Q} y(q) \cdot \Delta_t(q))} \in O(|C_0|)$$

occurrences of transitions that are in D.

Now we need to argue, that $C_0 \vDash F(\textsc{Dead}(D))$. For sake of contradiction assume that $C_0 \nvDash F(\textsc{Dead}(D))$. Let $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} \cdots$ be a fair execution starting at C_0. As π is infinite and there are only finitely many different configurations, there must be a configuration C_i that is visited infinitely often. By assumption and the definition of DEAD(D), there must be a configuration C_i' and transition $t \in D$ such that $C_i \xrightarrow{*} C_i' \xrightarrow{t} C_i''$. Thus, $C_0 \xrightarrow{*} C_i'$. Because of fairness and Lemma 2.0.1, π will take the step $C_i' \xrightarrow{t} C_i''$ infinitely often. This is a contradiction because there can only be finitely many occurrences of transitions that are in D. □

Note that the system of equations of Lemma 4.4.1 has a solution for some set D if and only if also has a solution $D' = \{t\}$ for every $t \in D$. Therefore, we simply test satisfiability of the constraints for singleton sets.

For a protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$, a vector $y \in \mathbb{Q}^{|Q|}$ and the sets of transitions

$D, T_{dead} \subseteq \mathcal{T}$ we define the following constraint:

$$\begin{aligned}\textsc{Ranking}(\mathcal{P}, y, D, T_{dead}) \stackrel{\text{def}}{=}\ & y \geq 0 \\ & \wedge \bigwedge_{t \in D} \sum_{q \in Q} y(q) \cdot \Delta_t(q) < 0 \\ & \wedge \bigwedge_{t \in (\mathcal{T} \setminus D)} \sum_{q \in Q} y(q) \cdot \Delta_t(q) \leq 0\end{aligned}$$

Listing 4.5 shows the implementation of EventuallyDead that uses the ranking function approach.

Listing 4.5: "Procedure: $\textsc{EventuallyDead}_{ranking}$"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)

result := T_dead

for each t ∈ T
    D := {t}
    if ∃y ∈ ℚ^|Q| : Ranking(P, y, D, T_dead)
        result = result ∪ {t}

return result
```

Deciding $\exists y \in \mathbb{Q}^{|Q|} : \textsc{Ranking}(\mathcal{P}, y, D, T_{dead})$ has complexity **P** because linear programming is in **P**. $\textsc{EventuallyDead}_{ranking}$ decides this constraint $|\mathcal{T}|$ times. Therefore, the procedure $\textsc{EventuallyDead}_{ranking}$ runs in polynomial time.

Layered Termination

In [5] Blodin *et al.* use *layered termination* to show termination of population protocols. The idea is to split the transitions of a protocol into layers such that transitions of a lower layer cannot reenable transitions of a higher layer. We reuse this approach to find a subset of transitions that is eventually dead.

Lemma 4.4.2. *Let $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ be a population protocol. Let $S = (T_{dead}, Q_{deserted})$ be a stage. Let $E = \mathcal{T} \setminus T_{dead}$ and $D \subseteq E$ be a set of transitions. Let y be a vector of unknowns*

containing one unknown $y(q)$ for every state $q \in \mathrm{Q}$. If the system of inequations

$$y \geq 0 \tag{4.18}$$

$$\sum_{q \in \mathrm{Q}} y(q) \cdot \Delta_t(q) < 0 \qquad \text{for all } t \in D \tag{4.19}$$

$$\bigvee_{u' \in T_{dead} \cup D} \mathrm{pre}(u') \leq \mathrm{pre}(t) + (\mathrm{pre}(u) \ominus \mathrm{post}(t)) \qquad \text{for all } t \in E \setminus D \text{ and } u \in D \tag{4.20}$$

has a rational solution, then $S \vDash F(\textsc{Dead}(D))$.

Proof. Assume that the system of inequations has a rational solution y. We define the ranking function $r(C) \stackrel{\text{def}}{=} \sum_{q \in \mathrm{Q}} y(q) \cdot C(q)$. As $y \geq 0$, $r(C) \geq 0$ for any configuration C. Because of (4.19) firing a transition $t \in D$ decreases the ranking.

First we show $\textsc{Dis}(D) \Rightarrow \textsc{Dead}(D)$. Assume that some transition $t \in E \setminus D$ can reenable some transition $u \in D$. Then there must some $C, C' \in S$ such that $C \xrightarrow{t} C'$ and $C \vDash \textsc{Dis}(D)$ and $C' \nvDash \textsc{Dis}(D)$. Therefore, $\mathrm{pre}(t) + (\mathrm{pre}(u) \ominus \mathrm{post}(t)) \leq C$. By 4.20 there is some $u' \in T_{dead} \cup D$ such that $\mathrm{pre}(u') \leq \mathrm{pre}(t) + (\mathrm{pre}(u) \ominus \mathrm{post}(t))$. This implies that u' was enabled in C. As T_{dead} are already dead by the definition of the stage, we know $u' \in D$. This is a contradiction to $C \vDash \textsc{Dis}(D)$.

We will now argue that for any $C \in S$ there exists a C' such that $C \xrightarrow{*} C'$ and $C' \vDash \textsc{Dis}(D)$. For sake of contradiction assume that C cannot reach such a configuration. We know $C \nvDash \textsc{Dis}(D)$. Therefore, there exists a transition $t \in D$ such that $C \xrightarrow{t} C'$. This decreases the ranking by at least

$$\min_{t \in D}(-\sum_{q \in \mathrm{Q}} y(q) \cdot \Delta_t(q)).$$

By assumption $C' \nvDash \textsc{Dis}(D)$. We can repeat the argument indefinitely. This would imply that we can decrease the ranking infinitely often by a constant. This is a contradiction to the fact that the ranking always is non-negative. Therefore, there must exist such a configuration C' such that $C \xrightarrow{*} C'$ and $C' \vDash \textsc{Dis}(D)$.

Next we show that any fair execution will eventually disable D. Let $C_0 \in S$ be a configuration and let $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2} C_2 \cdots$ be a fair execution. We necessarily have $t_i \in E$ for all $i \geq 0$. As π is infinite and there are only finitely many different configurations, there must be a configuration C that is visited infinitely often. As shown before, there must exist a C' such that $C \xrightarrow{*} C'$ and $C' \vDash \textsc{Dis}(D)$. Using fairness and Lemma 2.0.1, the execution will eventually disable D. As argued above, this implies that D is dead. Therefore, $S \vDash F(\textsc{Dead}(D))$. □

The satisfiability of the constraints of Lemma 4.4.2 depends on the actual set D (i.e. we cannot simply check all singleton sets). Therefore, the algorithm non-deterministically guesses a set $D \subseteq E$ and tests satisfiability of the constraints.

For a protocol $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$, a vector $y \in \mathbb{Q}^{|Q|}$ and the sets of transitions $D, T_{dead} \subseteq \mathcal{T}$ we define the following constraint:

$$\begin{aligned}\textsc{Layered}(\mathcal{P}, y, D, T_{dead}) \overset{\text{def}}{=}\ & y \geq 0 \\ & \wedge \bigwedge_{t \in D} \sum_{q \in Q} y(q) \cdot \Delta_t(q) < 0 \\ & \wedge \bigwedge_{t \in (\mathcal{T} \setminus T_{dead} \setminus D)} \ \bigwedge_{u \in D} \ \bigvee_{u' \in (T_{dead} \cup D)} \\ & \qquad \text{pre}(u') \leq \text{pre}(t) + (\text{pre}(u) \ominus \text{post}(t))\end{aligned}$$

Listing 4.6 shows the implementation of the layered termination approach for EventuallyDead.

Listing 4.6: "Procedure: $\textsc{EventuallyDead}_{layered}$"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)

if ∃D ⊆ (T − T_dead) : ∃y ∈ Q^|Q| : Layered(P, y, D, T_dead)
    return max_|·| {D ⊆ (T − T_dead) | ∃y ∈ Q^|Q| : Layered(P, y, D, T_dead)}
else return T_dead
```

Deciding $\exists y \in \mathbb{Q}^{|Q|} : \textsc{Layered}(\mathcal{P}, y, D, T_{dead})$ has complexity **P** because linear programming is in **P**. Because $\textsc{EventuallyDead}_{layered}$ needs to guess D, finding eventually dead transitions with this approach has complexity **NP**.

Combined Approach

The implementation using ranking functions and the implementation using layered termination are in general incomparable with respect to the set of transitions for with the constraints hold.

The protocols $\mathcal{P}_1$ and $\mathcal{P}_2$ of Figure 4.2 shows there are eventually dead transitions that can only be found by one of the approaches.

$\mathcal{P}_1$ is a part of the Flock-of-Birds protocol $\mathcal{P}_{X \geq 3}$ (see Figure 2.2). In this case, we can use the ranking function $y(C) = C(q_1) + 2 \cdot C(q_2)$ to show that t_{12} is eventually dead. Layered termination is not able to show this as $\textsc{Dis}(t_{12}) \not\Rightarrow \textsc{Dead}(t_{12})$.

$\mathcal{P}_2$ is part of the majority protocol $\mathcal{P}_{majority}$ (see Figure 2.1). For $\mathcal{P}_2$ we can use layered termination to show that t_{Ab} is eventually dead. For the set $D = \{t_{Ab}\}$ there is a suitable ranking function $y(C) = C(b)$. Once t_{Ab} is disabled then it is also dead (i.e. condition 4.20). By Lemma 4.4.2 t_{Ab} is eventually dead. The ranking function approach cannot

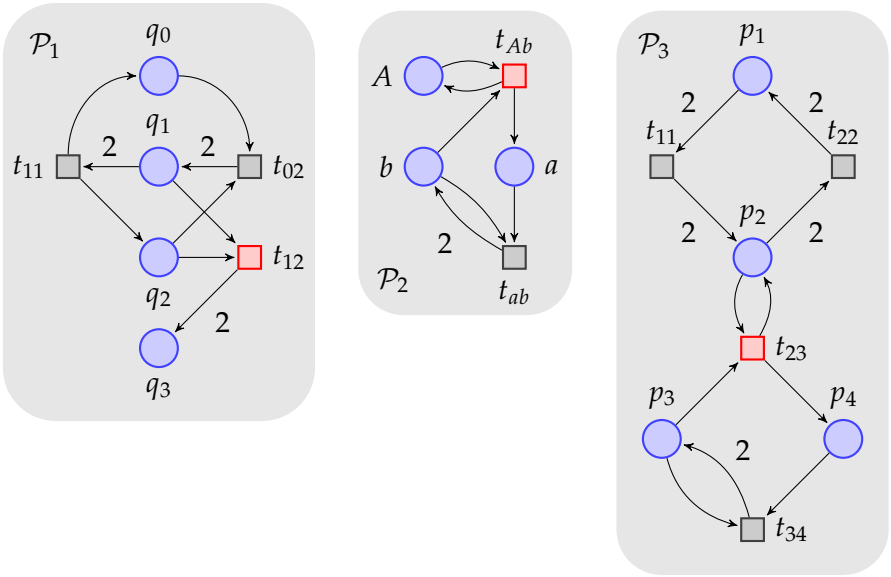


Figure 4.2.: Three population protocols with eventually dead transitions (in red).

show that t_{Ab} is eventually dead. It holds that $-\Delta_{t_{Ab}} = \Delta_{t_{ab}}$. Therefore, conditions 4.16 and 4.17 cannot both be true so that there is no suitable ranking function.

Thus, we add another implementation that solves both sets of constraints separately and combines the results. Listing 4.7 shows the implementation of the combined approach for EventuallyDead. Finding eventually dead transitions with the combined approach has complexity **NP**

Listing 4.7: "Procedure: $\textsc{EventuallyDead}_{combined}$"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)

return EventuallyDead_ranking(P, φ_pre, S) ∪ EventuallyDead_layered(P, φ_pre, S)
```

Note that even the combined approach is not complete because it cannot show that transition t_{23} of protocol $\mathcal{P}_3$ in Figure 4.2 is eventually dead.

4.4.2. Terminal

As we only have an overapproximation for the set of configurations in the stage S, we define:

$$\textsc{Terminal}(\mathcal{P}, \varphi_{pre}, S, \varphi_{post}) \stackrel{\text{def}}{=} \neg(\exists C \vDash (\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \wedge \neg\varphi_{post}))$$

This implementation satisfies the requirements given in Section 4.2:

$$\begin{aligned}
&\textsc{Terminal}(\mathcal{P}, \varphi_{pre}, S, \varphi_{post}) \\
\Leftrightarrow\quad & \neg(\exists C \vDash (\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \wedge \neg\varphi_{post})) \\
\Leftrightarrow\quad & \forall C : C \vDash \neg\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \vee \varphi_{post} \\
\Rightarrow\quad & \forall C : C \notin S \vee \varphi_{post} && \text{(by def. of PotInStage)} \\
\Leftrightarrow\quad & \forall C \in S : C \vDash \varphi_{post} \\
\Leftrightarrow\quad & S \vDash \varphi_{post}
\end{aligned}$$

Under the assumption that φ_{post} is quantifier free, the complexity of deciding the constraint $\textsc{Terminal}(\mathcal{P}, \varphi_{pre}, S, \varphi_{post})$ is **co-NP**. The size of $\textsc{Terminal}(\mathcal{P}, \varphi_{pre}, S, \varphi_{post})$ depends on the overapproximation of the stage that is used: $|\textsc{PotInStage}| + |\varphi_{post}|$

4.4.3. Split

We need to split a stage $S = (T_{dead}, Q_{deserted})$ into stages $S_1, S_2, \cdots$ with more deserted states. If a state is part of some empty siphon, it is deserted. Thus, we try to compute a set of siphons $\mathcal{R}$ such that every configuration of the stage S leaves at least one siphon $R \in \mathcal{R}$ empty. We define:

$$\begin{aligned}
\textsc{UsefulSiphon}(\mathcal{P}, \varphi_{pre}, S, \mathcal{R}, Q_{siphon}) \stackrel{\text{def}}{=} \exists C :\ & \\
& \textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, C) \\
& \wedge \textsc{LargestEmptySiphon}(\mathcal{P}, C, Q_{siphon}) \\
& \wedge \bigwedge_{R \in \mathcal{R}} C \vDash \neg\textsc{Empty}(R)
\end{aligned}$$

Intuitively, $\mathcal{R}$ is a set of previously found useful siphons. A set of states Q_{siphon} is a useful siphon if it is the largest empty siphon of a configuration $C \in S$ such that in C all previously found siphons are not empty. The implementation for Split can be seen in Listing 4.8.

Listing 4.8: "Procedure: Split"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)

R := ∅

while (∃ Q_siphon ⊆ Q : UsefulSiphon(P, φ_pre, S, R, Q_Siphon))
    U := min_|·| { Q_siphon ⊆ Q | UsefulSiphon(P, φ_pre, S, R, Q_Siphon) }
    if Q_deserted ⊂ U
        R = R ∪ {U}
    else
        fail

return { (T_dead ∪ { t ∈ T | •t ∩ Q_siphon ≠ ∅ }, Q_siphon) | Q_siphon ∈ R }
```

Lemma 4.4.3. *Let $\mathcal{P} = (Q, T, \Sigma, I, O)$ be a population protocol, let φ_{pre} be a precondition and let $S = (T_{dead}, Q_{deserted})$ be a stage. If a successful execution of the procedure in Listing 4.8 produces the substages $S_1, \cdots, S_k$ where $S_i = (T_{dead_i}, Q_{deserted_i})$ then*

$$\left(\bigwedge_{i=1}^{k} Q_{deserted} \subset Q_{deserted_i}\right) \wedge \left(S \in \bigcup_{i=1}^{k} S_i\right)$$

Proof. Because the procedure only adds a siphon U if $Q_{deserted} \subset U$, we know $Q_{deserted} \subset Q_{deserted_i}$ for all $1 \leq i \leq k$.

For sake of contradiction assume there is a $C \in S$ such that $C \notin \bigcup_{i=1}^{k} S_i$. We know $\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, C)$ by the definition of PotInStage. If $C \vDash \textsc{Empty}(Q_{deserted_i})$ for some $1 \leq i \leq k$, then $C \vDash \textsc{Deserted}(Q_{deserted_i})$ as $Q_{deserted_i}$ is a siphon. This would imply that the transitions $\{t \in T \mid {}^\bullet t \cap Q_{deserted_i} \neq \varnothing\}$ are dead in C such that $C \in S_i$. Therefore, $C \vDash \bigwedge_{i=1}^{k} \neg\textsc{Empty}(Q_{deserted_i})$. Let $Q_{siphon,C}$ be the largest empty siphon in C. Thus, $\textsc{LargestEmptySiphon}(\mathcal{P}, C, Q_{siphon,C})$. By the definition of UsefulSiphon, we know $\textsc{UsefulSiphon}(\mathcal{P}, \varphi_{pre}, S, \{Q_{deserted_i} \mid 1 \leq i \leq k\}, Q_{siphon,C})$ is *true*. This is a contradiction to the fact that the procedure Split ended and was successful. □

In general, one can choose any empty siphon for a split. We however always choose a largest empty siphon with as few states as possible. This heuristic keeps the number of substages small. To illustrate this, we will compare our heuristic to the heuristic that always chooses the largest empty siphon with the most states. Consider the protocol of Figure 4.3 with the precondition that the states $q_1, \cdots, q_n$ start with exactly one agent and

the state *init* starts with $k \leq n$ agents. Let S be the stage where the transitions $t_1, \cdots, t_n$ are dead. Note that any $P \cup \{init\}$ with $P \subseteq \{q_1, \cdots, q_n\}$ is the largest empty siphon in some configuration $C \in S$.

Let Q, Q' be two siphons such that $Q \subset Q'$. If we first split with Q, we do not need to split with Q' afterwards because whenever Q' is empty, so is Q. However, if we split with Q' first, then we need to split with Q afterwards because it is possible that Q is empty while Q' is not empty.

Therefore, splitting S with our heuristic results in just one substage where *init* is deserted. If we always choose the largest empty siphon with the most states first, we will produce $O(2^n)$ substages.

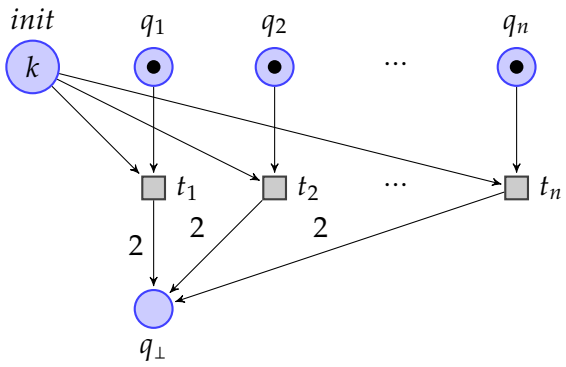


Figure 4.3.: Splitting heuristic: Example protocol 1.

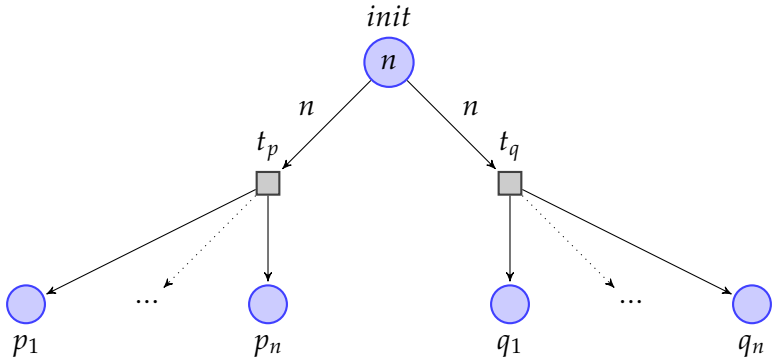


Figure 4.4.: Splitting heuristic: Example protocol 2.

Another reason to use our heuristic can be found in Figure 4.4. Consider the case where there are initially n agents in state *init*. We want to split the stage where t_p and t_q are dead. Our heuristic results in exactly two substages by choosing the two siphons

$\{p_1, \cdots, p_n\}$ and $\{q_1, \cdots, q_n\}$. This is the minimal number of substages. Other heuristics might split these two siphons further, resulting in more substages.

We will now analyse the complexity of Split. Assume the implementation already found the set of useful siphons $\mathcal{R}$. Finding a new useful siphon corresponds to deciding $\exists Q_{siphon} \subseteq \mathrm{Q} : \textsc{UsefulSiphon}(\mathcal{P}, \varphi_{pre}, S, \mathcal{R}, Q_{Siphon})$. This is an $\exists$-Presburger constraint implying a complexity of **NP**. There could be an exponential amount of substages. However, in Section 7.1.2 we see that the number of produced substages is small. Note that the size of the constraint $\exists Q_{siphon} \subseteq \mathrm{Q} : \textsc{UsefulSiphon}(\mathcal{P}, \varphi_{pre}, S, \mathcal{R}, Q_{Siphon})$ depends on the overapproximation of the stage that is used: $|\textsc{PotInStage}| + |\mathcal{R}|$.

4.5. Example

Example 1. An automatically computed stage graph for the protocol $\mathcal{P}_{majority}$ can be seen in Figure 4.5. Edges leading to substages that resulted from an EventuallyDead call are labeled with the approach and the ranking function that were used. All other edges correspond to successful Split calls.

In stage S_1, the transition t_{AB} is dead. Thus, in any configuration $C \in S_1$, either $C(A) = 0$ or $C(B) = 0$. As $\{A\}$ and $\{B\}$ are siphons, the algorithm successfully splits into two substages.

In stage S_2, all transitions but t_{Ba} and t_{ab} are dead. Both of them reduce the number of agents in state a when fired. Thus, t_{Ba} and t_{ab} are eventually dead using the ranking function approach.

In stage S_5, the algorithm shows that t_{Ab} is eventually dead using the layered termination approach. This is possible because once the transition t_{Ab} is disabled, it is dead. There are two cases: Either there is no agent in state A. Then t_{Ab} is dead because A is deserted. Otherwise, there is no agent in state b. In that case all transitions are disabled. As there are no transitions possible that enable t_{Ab}, the transition must be dead.

Example 2. Figure 4.6 shows the two automatically computed stage graphs for the correctness of the protocol $\mathcal{P}_{X \geq 3}$ of Figure 2.2. The red stage graph verifies the case $X < 3$ and the blue stage graph verifies the case $X \geq 3$. The pre- and postconditions are:

$$\varphi_{pre_{X<3}} \stackrel{\text{def}}{=} q_0 < 3 \qquad \varphi_{post_{X<3}} \stackrel{\text{def}}{=} q_3 = 0$$
$$\varphi_{pre_{X\geq 3}} \stackrel{\text{def}}{=} q_0 \geq 3 \qquad \varphi_{post_{X\geq 3}} \stackrel{\text{def}}{=} q_0 = 0 \wedge q_1 = 0 \wedge q_2 = 0$$

In the case $X < 3$, one can instantly show the consensus *false* using the terminal check. In the case $X \geq 3$, the algorithm first shows that the transitions t_{03}, t_{13} and t_{23} are

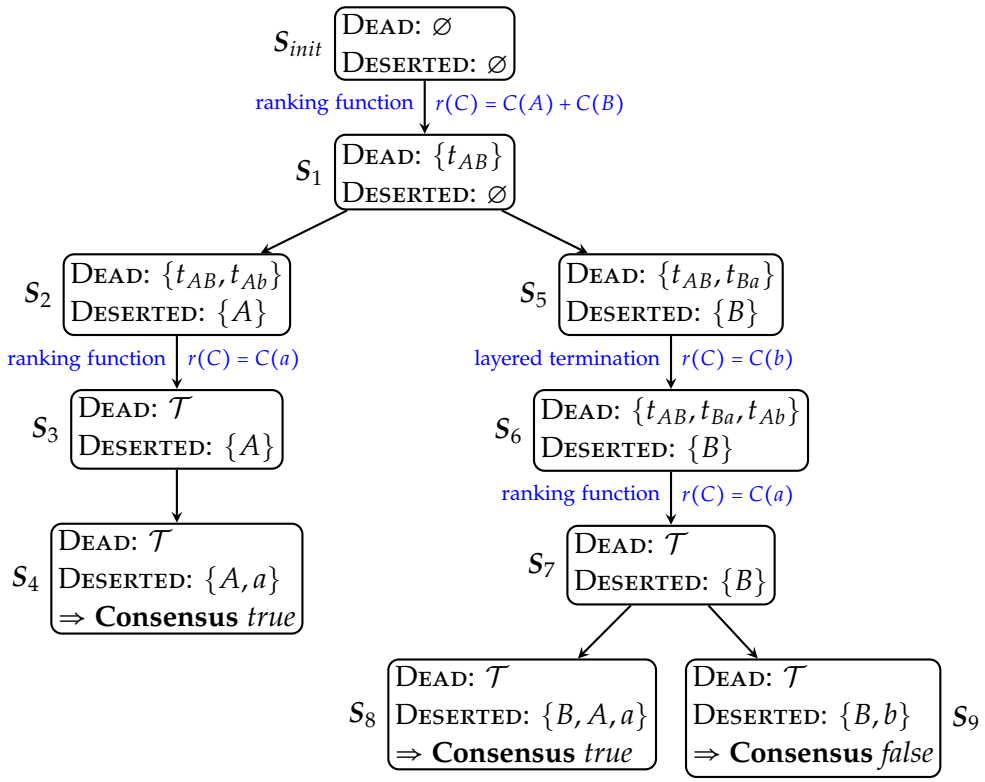


Figure 4.5.: Automatically generated stage graph for consensus of $\mathcal{P}_{majority}$.

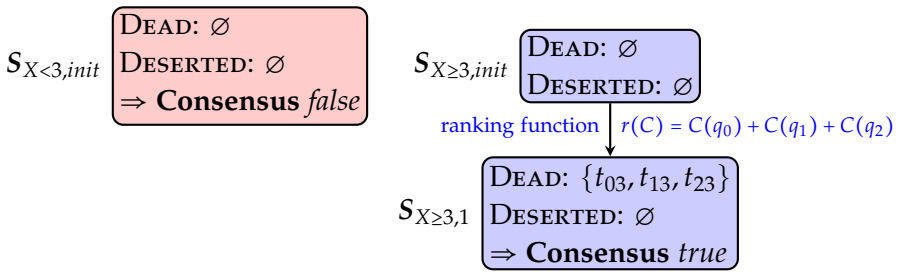


Figure 4.6.: Automatically generated stage graph for correctness of $\mathcal{P}_{X\geq 3}$.

eventually dead using the ranking function approach. Afterwards, it can show the consensus *true*.

4.6. Short Comparison with Approach for silent Protocols

The algorithm by Blondin *et al.* [5] for the verification of silent population protocols consists of two parts. The first part uses layered termination and tries to show that all transitions are eventually dead. The second part tries to show that every terminal configuration potentially reachable from some initial configuration has the correct output.

Although there are some similarities like the notions of layered termination and potential reachability, their algorithm uses the fact that all transitions are eventually dead. Note that it is rather easy to describe this kind of termination as there is a simple characterization of terminal configurations:

$$\mathit{Terminal}(C) \Leftrightarrow \bigwedge_{t \in \mathcal{T}} \bigvee_{q \in {}^\bullet t} C(q) < \mathrm{pre}(t)(q)$$

For non-silent population protocols, termination behaviour can be more complicated. A lasting consensus is not as easy to describe as the configuration may still change. This gives rise to the idea of a terminal stage in our stage graph.

We will now argue that our approach is a more general version of the algorithm for silent protocols. Assume a protocol can be verified by the algorithm for silent protocols. Therefore, it is possible to show that all transitions are eventually dead using layered termination. As our algorithm also uses layered termination, it can show that all transitions are eventually dead. This results in a stage graph where all terminal stages only contain terminal configurations. Afterwards, both algorithms use potential reachability to show that all reachable terminal configurations have the right output. Thus, the algorithm by Blondin *et al.* is to some degree a special case of our algorithm.

5. Termination Time

In practice, we are often not only interested in the correctness of a protocol but also want that a protocol terminates fast. Thus, we adapt the approach from [8] where Blondin *et al.* use a closely related version of stage graphs to automatically bound the speed of a protocol.

Up to this point, we used a non-probabilistic model together with a fairness assumption when arguing about executions of population protocols. However, in this section, we switch to a probabilistic model where the next pair of agents that interact are chosen uniformly at random from all ordered pairs. Note that this model is only defined for 2-way protocols. Thus, we only consider 2-way protocols in this section. It is easy to see that every execution in the probabilistic model is fair with probability 1.

The new model allows us to describe the expected number of interactions until a protocol terminates (i.e. until $G\varphi_{post}$ holds for some postcondition φ_{post}). We call the expected number of interactions until a protocol $\mathcal{P}$ terminates the *speed* of $\mathcal{P}$.

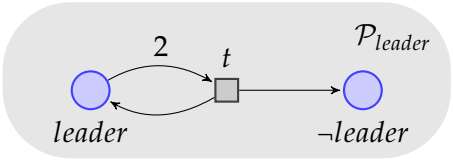


Figure 5.1.: Protocol $\mathcal{P}_{leader}$ for leader election.

Figure 5.1 shows a simple protocol for leader election where each agent can either be a leader or a non-leader. As an example, we will now compute the speed of the leader election process when every agent starts as a leader. In other words: We are interested in the expected number of interactions until there is exactly one leader left. Note that we always assume that any pair of agents can interact. Thus, if there are n agents and $1 < l \leq n$ leaders, then the probability that two leaders meet is $\frac{l}{n} \cdot \frac{l-1}{n-1}$. Therefore, the

expected number of interactions until termination is

$$\begin{aligned}\sum_{l=2}^{n} \frac{n}{l} \cdot \frac{n-1}{l-1} &= n(n-1) \cdot \sum_{l=1}^{n-1} \frac{1}{l(l+1)} \\ &= n(n-1) \cdot \sum_{l=1}^{n-1} \left(\frac{1}{l} - \frac{1}{l+1} \right) \\ &= n(n-1) \cdot \left(1 - \frac{1}{n} \right) \\ &= O(n^2).\end{aligned}$$

We will extend the algorithm given in Section 4.4 such that for each non-terminal stage we additionally get an upper bound on the expected number of interactions needed to enter some substage. This is typically a function that depends on the size of the initial configuration n. We call this function the *speed* of the stage. The asymptotic maximum of all speeds in a stage graph is an upper bound for the expected number of interactions in the whole protocol and is called the *speed* of the stage graph.

5.1. Split

If the substages of a stage S resulted from a successful Split call then every configuration $C \in S$ is already part of a substage by definition of Split. Thus, the speed of S is 0.

5.2. AlmostSurelyDead

We add a new implementation of EventuallyDead that returns eventually dead transitions T_{dead} and an upper bound for the expected number of interactions until Dead(T_{dead}) (see Listing 5.1). The computed upper bound is parametric in n where n is the size of the initial configuration (i.e. the number of agents).

Listing 5.1: "Procedure: EventuallyDead$_{time}$"

```
input: protocol 𝒫 = (Q, 𝒯, Σ, ℐ, 𝒪)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)

R := EventuallyDead_ranking(𝒫, φ_pre, S)
if R ≠ ∅
    F := FixPoint(𝒫, φ_pre, S, R)
    if F ≠ ∅
        if Fast(𝒫, φ_pre, S, F)
            return F, "O(n² log n)"
        else
            return F, "O(n³)"
    else
        return R, "O(n^c)"
else
    L := EventuallyDead_layered(𝒫, φ_pre, S)
    if R ≠ ∅
        return L, "e^{O(n log n)}"
    else
        return ∅, "no bound"
```

5.2.1. Layered Termination

If we find eventually dead transitions using the layered termination approach, we can guarantee a speed of $e^{O(n \log n)}$.

Lemma 5.2.1. *Let $\mathcal{P}$ be a population protocol with precondition φ_{pre} and let S be a stage. Let $L = \textsc{EventuallyDead}_{layered}(\mathcal{P}, \varphi_{pre}, S)$. For any configuration $C \in S$ of size $n = |C|$ the expected number of interactions until $\textsc{Dead}(L)$ is $e^{O(n \log n)}$.*

Proof. Let y be a solution vector for the conditions in Lemma 4.4.2. Recall the definition of the ranking function $r(C) \stackrel{\text{def}}{=} \sum_{q \in Q} y(q) \cdot C(q)$. As $y \geq 0$, $r(C) \geq 0$ for any configuration C. Because of (4.19) firing a transition $t \in L$ decreases the ranking by at least the constant

$$\alpha \stackrel{\text{def}}{=} \min_{t \in L}(-\sum_{q \in Q} y(q) \cdot \Delta_t(q)).$$

We define the constant

$$\beta \stackrel{\text{def}}{=} \max_{q \in Q}(y(q)).$$

Furthermore, we know that $R(C_0) \leq n \cdot \beta$. Thus, for any $C \in S$ there always is sequence $\pi \in L^*$ of length at most $\frac{\beta}{\alpha} n \in O(n)$ leading to a configuration C' such that $C' \vDash Dis(D)$. As shown in Lemma 4.4.2, this implies $C' \vDash \textsc{Dead}(L)$. The probability of firing a given transition is at least $1/n^2$. Thus, the probability of firing π is at least $n^{-2O(n)} = e^{-O(n \log n)}$. Therefore, the expected number of interactions until the transitions L are dead is $e^{O(n \log n)}$. □

5.2.2. Ranking Function

If we find eventually dead transitions using the ranking function approach, we can guarantee a speed of $O(n^c)$ for some constant c that only depends on the protocol but not the input.

Lemma 5.2.2. *Let $\mathcal{P}$ be a population protocol with precondition φ_{pre} and let S be a stage. Let $R = \textsc{EventuallyDead}_{ranking}(\mathcal{P}, \varphi_{pre}, S)$. For any configuration $C \in S$ of size $n = |C|$ the expected number of interactions until $\textsc{Dead}(R)$ is $O(n^c)$ for some constant c that only depends on $\mathcal{P}$.*

Proof. Let $C \in S$ be a configuration. If a fair run reaches a configuration $C' \vDash \textsc{Dis}(R)$, then either Dead(R) or R is only temporarily disabled. If R is only temporarily disabled, then there is a C'' such that $C' \xrightarrow{*} C''$ and $C'' \vDash \neg\textsc{Dis}(R)$.

We will now argue that C'' is reachable from C' in at most d transitions, where d is a constant depending only on the protocol. The set of configurations that can enable some transition $t \in R$ is upward closed with respect to point-wise ordering. Thus, there are finitely many minimal configurations with this property, that can be computed using the backwards coverability algorithm [11][12] (see Listing 4.4). Each of the minimal configurations can enable some $t \in R$ in a constant number of transitions and all larger configurations can enable t with the same sequence. Therefore, we can choose d as the maximum of those finitely many computable constants.

The probability of executing some transition $t \in R$ in at most $d+1$ steps is at least $n^{-2(d+1)}$. Hence, on average, we need to perform at most $n^{2(d+1)}$ sequences of length $d+1$ to fire some transition in R. Using Lemma 4.4.1, we know that any run starting at C contains at most $O(n)$ occurrences of transitions in R. We conclude that the expected number of interactions until $\textsc{Dead}(R)$ is $O(n) \cdot n^{2(d+1)} \cdot (d+1)$ or $O(n^c)$ for $c = 2d+3$. □

5.2.3. FixPoint

The so called "fixpoint" algorithm is applied after $\textsc{EventuallyDead}_{ranking}$ if the call returned a non-empty set R of transitions. It computes a subset $F \subseteq R$ of transitions such that $\textsc{Dis}(F) \Rightarrow \textsc{Dead}(F)$. Intuitively, this combines the constraints of the ranking function approach and the layered termination approach.

Listing 5.2: "Procedure: FixPoint"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)
       transitions R (⊆ T)          # eventually dead

F := R

while ∃u ∈ F : ∃t ∈ (T ∖ T_dead ∖ F) :
                 ⋀_{u'∈T_dead∪F} pre(u') ≰ pre(t) + (pre(u) ⊖ post(t))
    F := F ∖ {u'}

return F
```

The while loop tests the condition 4.20 of layered termination and iteratively removes transitions that violate $\textsc{Dis}(F) \Rightarrow \textsc{Dead}(F)$. The complexity of FixPoint is $POLY(|\mathcal{T}|)$.

Lemma 5.2.3. *Let $\mathcal{P}$ be a population protocol with precondition φ_{pre} and let S be a stage. Let $R = \textsc{EventuallyDead}_{ranking}(\mathcal{P}, \varphi_{pre}, S)$. Let $F = \textsc{FixPoint}(\mathcal{P}, \varphi_{pre}, S, R)$. For any configuration $C \in S$ of size $n = |C|$ the expected number of interactions until $\textsc{Dead}(F)$ is $O(n^3)$.*

Proof. Observe that $F \subseteq R$. Because of Lemma 4.4.1 we know that any run starting at C contains $O(n)$ occurrences of transitions in F.

If $\neg\textsc{Dis}(F)$ the probability of executing some transition in F is at least $1/n^2$. Therefore, the expected number of interactions until $\textsc{Dis}(F)$ is $O(n) \cdot n^2 = O(n^3)$.

The FixPoint algorithm removes transitions where the negation of condition 4.20 holds. Therefore, condition 4.20 holds for F. As argued in the proof of Lemma 4.4.2 this implies $\textsc{Dis}(F) \Rightarrow \textsc{Dead}(R)$. □

5.2.4. Fast

Let y be a solution to the constraint $\textsc{Ranking}(\mathcal{P}, y, F, T_{dead})$. Then $r(C) = \sum_{q \in Q} y(q) \cdot C(q)$ is the ranking function that proves that T_{dead} is eventually dead.

After a successful call of FixPoint we can improve the speed bound to $O(n^2 \log n)$ if we can guarantee that for every $q \in Q$ with $y(q) > 0$ and $C \in S$ such that $C(q) > 0$ and $C \notin \textsc{Dis}(F)$ there exists a $t \in F$ such that $q \in {}^\bullet t$ and $\textsc{Enabled}(t)$.

Intuitively, for every state q that impacts the ranking function, there always must be at least one enabled transition that involves q and reduces the ranking function. Or in other words, every agent that impacts the ranking can always reduce the ranking. If this holds, then we call S *fast*. Note that there might be multiple possible ranking functions. Thus, we check whether any ranking function can be used to show that S is fast.

We define PotSlow for a vector y, a set of transitions F and a state q:

$$\textsc{PotSlow}(\mathcal{P}, \varphi_{pre}, S, y, F, q) \stackrel{\text{def}}{=} \exists C : C \vDash \textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \wedge q > 0 \wedge \left(\bigvee_{t \in F} \textsc{Enabled}(t)\right) \wedge \bigwedge_{t \in F} (q \notin {}^\bullet t \vee \textsc{Dis}(t))$$

Listing 5.3: "Procedure: Fast"

```
input: protocol P = (Q, T, Σ, I, O)
       Presburger predicate φ_pre
       stage S = (T_dead, Q_deserted)
       transitions F (⊆ T)          # eventually dead

failed := ∅

while ∃y : Ranking(P, y, F, T_dead) ∧ ∧_{q∈failed} y(q) = 0
    fast := True
    for each q ∈ Q with y(q) > 0
        if PotSlow(P, φ_pre, S, y, q)
            failed := failed ∪ {q}
            fast := False
            break
    if fast
        return True

return False
```

Deciding PotSlow has complexity **NP** and the size of the constraint depends on the overapproximation for the stage that was used. An execution of Fast might need to solve $O(|Q|)$ linear programs to find ranking functions and decide the constraint PotSlow up to $O(|Q|^2)$ times.

Lemma 5.2.4. *Let $\mathcal{P}$ be a population protocol with precondition φ_{pre} and let S be a stage. Let $R = \textsc{EventuallyDead}_{ranking}(\mathcal{P}, \varphi_{pre}, S)$. Let $F = \textsc{FixPoint}(\mathcal{P}, \varphi_{pre}, S, R)$. If the procedure $\textsc{Fast}(\mathcal{P}, \varphi_{pre}, S, F)$ returns true, then for any configuration $C \in S$ of size $n = |C|$ the expected number of interactions until $\textsc{Dead}(F)$ is $O(n^2 \log n)$.*

Proof. Let y be the solution to $\textsc{Ranking}(\mathcal{P}, y, F, T_{dead})$ where Fast returned *true*. Using the definitions of PotSlow and PotInStage, we know that for all $q \in Q$ with $y(q) > 0$:

$$\begin{aligned}
&\neg\textsc{PotSlow}(\mathcal{P}, \varphi_{pre}, S, y, q)\\
&\Leftrightarrow \forall C : C \vDash \left(\neg\textsc{PotInStage}(\mathcal{P}, \varphi_{pre}, S) \vee q = 0 \vee \textsc{Dis}(F) \vee (\exists t \in F : q \in {}^\bullet t \wedge \neg\textsc{Dis}(t))\right)\\
&\Rightarrow S \vDash (q = 0 \vee \textsc{Dis}(F) \vee (\exists t \in F : q \in {}^\bullet t \wedge \neg\textsc{Dis}(t)))
\end{aligned}$$

We define the ranking function $r(C) \stackrel{\text{def}}{=} \sum_{q \in Q} y(q) \cdot C(q)$. As $y \geq 0$, $r(C) \geq 0$ for any configuration C. Using the definition of Ranking, we know that no transition increases the ranking. Firing a transition $t \in F$ decreases the ranking by at least the constant

$$\alpha \stackrel{\text{def}}{=} \min_{t \in F}(-\sum_{q \in Q} y(q) \cdot \Delta_t(q)).$$

We define the constant

$$\beta \stackrel{\text{def}}{=} \max_{q \in Q}(y(q)).$$

By the definition of $r(C)$, we know that $r(C) \leq n \cdot \beta$. If $r(C) = 0$, then we cannot decrease the ranking anymore and therefore $C \vDash \textsc{Dead}(F)$. Thus, in any run starting at any $C \in S$ there are at most $\frac{\beta}{\alpha} n = O(n)$ transitions that are in F.

We claim that the expected number of interactions until $Dis(F)$ or some $t \in F$ is fired is bounded by $\gamma \frac{n^2}{r(C)}$ for some constant γ that only depends on $\mathcal{P}$. We use this to calculate the expected number of interactions until $\textsc{Dis}(F)$. In the worst case, we need to fire $\frac{\beta}{\alpha} n$ transitions in F. Each time the ranking is reduced by at least α. Therefore, the expected number of interactions until $\textsc{Dis}(F)$ is at most:

$$\sum_{i=1}^{\frac{\beta}{\alpha} n} \gamma \frac{n^2}{\alpha \cdot i} = \frac{\gamma}{\alpha} n^2 \sum_{i=1}^{\frac{\beta}{\alpha} n} \frac{1}{i} = O(n^2 \log n)$$

As F was the result of the FixPoint algorithm, we know that $\textsc{Dis}(F) \Rightarrow \textsc{Dead}(F)$.

Now we show the claim. Let $C \in S$ such that $C \vDash \neg\textsc{Dis}(F)$. Using the pigeonhole principle and the definition of the ranking function, we know that there must a state $A \in Q$ such that $C(A) \cdot y(A) \geq r(C)/|Q|$. As it is possible to fire a $t \in F$, we know $r(C) > 0$. Thus, $C(A) > 0$ and $y(A) > 0$. Therefore, we know

$$C(A) \geq \frac{r(C)}{|Q| \cdot y(A)} \geq \frac{r(C)}{|Q| \cdot \beta}$$

and

$$\exists t \in F : q \in {}^\bullet t \wedge \neg \textsc{Dis}(t).$$

The probability of firing t in C is at least $\frac{C(A)}{n} \cdot \frac{1}{n} \geq \frac{1}{\beta|Q|} \cdot \frac{r(C)}{n^2}$. If this trail is *unsuccessful*, we reach a configuration C'. If $C' \models \neg Dis(F)$, another *independent* trial with the same probabilities is performed in C'. Thus, the expected number of interactions until $Dis(F)$ or some $t \in F$ is fired is bounded by $\beta|Q| \cdot \frac{n^2}{r(C)}$. □

5.3. Example

If we compute the stage graph for consensus of $\mathcal{P}_{majority}$ with this method, we get the upper bounds of Figure 5.2. Therefore, the overall speed of the protocol is $e^{O(n \log n)}$ as this is the asymptotic maximum of all bounds.

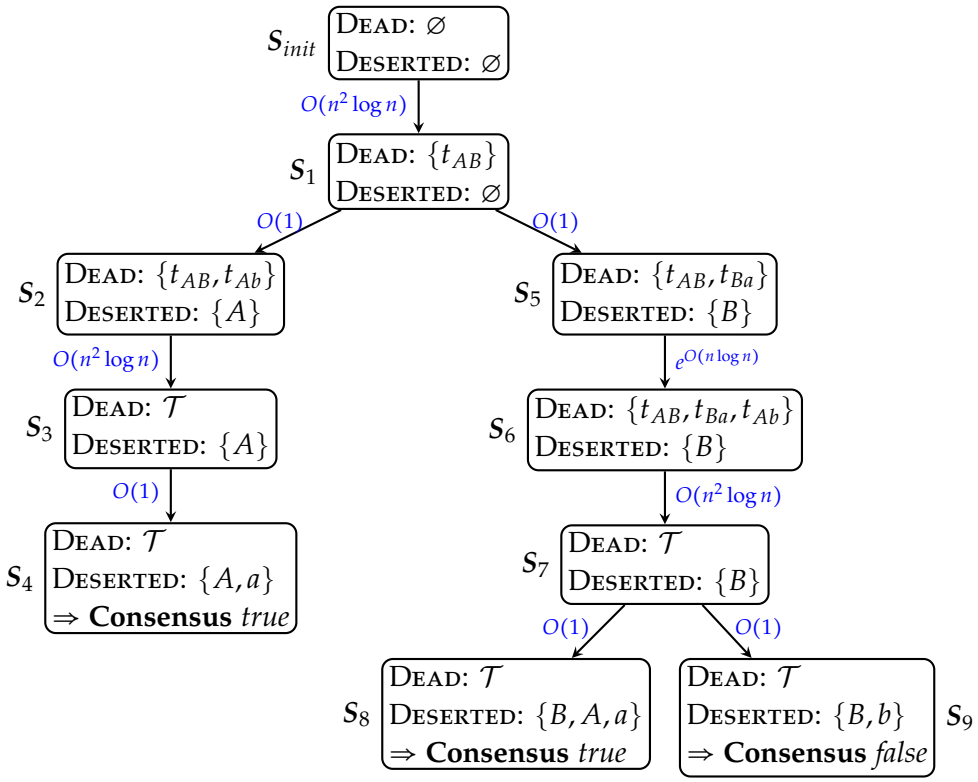


Figure 5.2.: Automatically generated stage graph with speed bounds for consensus of $\mathcal{P}_{majority}$.

We will now examine the transition from S_2 to S_3 and argue that this transition is fast. Let $D = \{t_{Ba}, t_{ab}\}$. First, we use the ranking function approach with $r(C) = C(a)$ to show that the transition has the speed $O(n^c)$ for some constant c (Lemma 5.2.2).

To show a speed of $O(n^3)$, we need to argue that $S_2 \vDash \textsc{Dis}(D) \Rightarrow \textsc{Dead}(D)$. This is true because all other transitions are already dead. Hence, if the transitions D are disabled, then all transitions are disabled.

Finally, we need to show the fast property. The only state that impacts the ranking $r(C)$ is the state a. Let $C \in (S_2 \setminus S_3)$ such that $C(a) > 0$. We know $C \vDash \neg\textsc{Dis}(D)$ because otherwise this would imply $C \vDash \textsc{Dead}(D)$ and $C \in S_3$. Thus, in C there is an enabled transition that involves a. This implies the speed of $O(n^2 \log n)$.

6. LTL

Using the algorithm of Section 4.3, we can verify postcondition properties of population protocols. In general, these can be expressed as $FG\varphi$ where φ is a Presburger predicate. This allows us to verify consensus or correctness as well as proof leader election. However, one can use population protocols to model a wide range of systems. In that case, we might want to verify other properties of the protocol like liveness, that have a different form. In this section, we show how to use stage graphs to verify properties such as liveness of a transition using stage graphs.

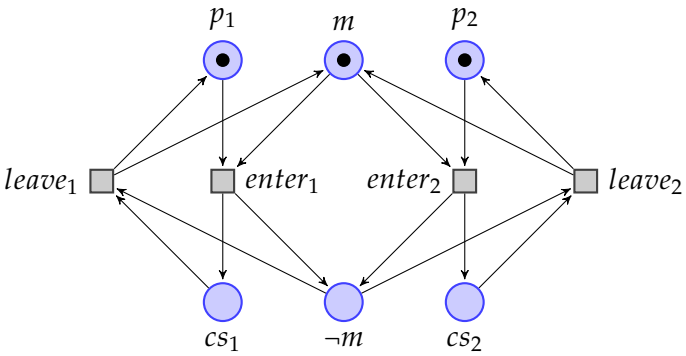


Figure 6.1.: Simple mutex population protocol $\mathcal{P}_{SimpleMutex}$ for two processes.

Let's say we have the simple mutex population protocol of Figure 6.1. A desirable liveness property is that process 1 enters its critical section infinitely often. This can be described using the following LTL formula: $GF(enter_1)$. Note that this is an LTL formula not over configurations (as defined before). Instead, the LTL formula is action-based (i.e. over transitions). We are not using a formula over states because model checking is undecidable for state-based LTL [14]. We need to define labels for the transitions so that we can refer to them in the formula.

Definition 4. *A* labeled population protocol *is a population protocol* $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ *together with a label function* $l : \mathcal{T} \to \Psi$ *that assigns a label of the alphabet* Ψ *to every transition.*

In this section, we introduce a construction that allows us to verify such LTL properties over labeled population protocols. We follow the standard automata-theoretic

approach [15]. The general idea is to first translate the negation of the LTL formula into a *limit-deterministic Büchi automaton* (LDBA) [16]. Then we perform a product construction of the Büchi automaton and the labeled protocol. This allows us to check that fair runs of the resulting protocol cannot fulfil the acceptance condition of the Büchi automaton.

In general, one would like to use a type of automaton for model checking that is as small as possible. Thus, the natural choice is a non-deterministic automaton like *NBA*. However, non-determinism complicates verification [17]. Therefore, we use LDBAs for our construction. They enable verification and can be more compact than deterministic automata because they allow a non-deterministic prefix.

6.1. Limit-Deterministic Büchi Automaton

Definition 5 (NBA)**.** *A non-deterministic Büchi automaton is a tuple* $B = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ *such that*

- Q *is a finite set of* states,
- Σ *is a finite* alphabet,
- $\delta : Q \times \Sigma \to \left(2^Q \setminus \{\varnothing\}\right)$ *is a* transition function,
- $\mathcal{I} \subseteq Q$ *is the set of* initial states *and*
- $\mathcal{F} \subseteq Q \times \Sigma \times Q$ *is the set of* accepting transitions.

An infinite word $\pi = a_1 a_2 \cdots$ *is accepted by* B *if and only if there is a corresponding infinite path* $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots$ *such that*

- $q_0 \in \mathcal{I}$,
- $q_{i+1} \in \delta(q_i, a_{i+1})$ *for all* $i \geq 0$, *and*
- *there are infinitely many* $i \geq 0$ *such that* $(q_i, a_{i+1}, q_{i+1}) \in \mathcal{F}$.

The language of B *is the set* $L(B)$ *containing all infinite words over* Σ *that are accepted by* B.

Definition 6 (LDBA)**.** *A limit-deterministic Büchi automaton is a Büchi automaton* $B = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$ *such that* Q *can be partitioned into two disjoint sets* $Q = Q_N \uplus Q_D$, *such that*

1. $\delta(q, a) \subseteq Q_D$ *and* $|\delta(q, a)| = 1$ *for all* $q \in Q_D, a \in \Sigma$, *and*
2. $\mathcal{F} \subseteq Q_D \times \Sigma \times Q_D$.

Intuitively, a limit-deterministic Büchi automaton has a non-deterministic component (Q_N) and a deterministic component (Q_D). All accepting transitions are in the deterministic component and once the automaton is in the deterministic component it must stay there forever. Therefore, there might be a non-deterministic prefix, but afterwards, the automaton behaves deterministically. Figure 6.2 shows an LDBA over the alphabet that corresponds to the transitions of $\mathcal{P}_{SimpleMutex}$. If the red transition is accepting, the automaton accepts the negation of the LTL formula $GF(enter_1)$.

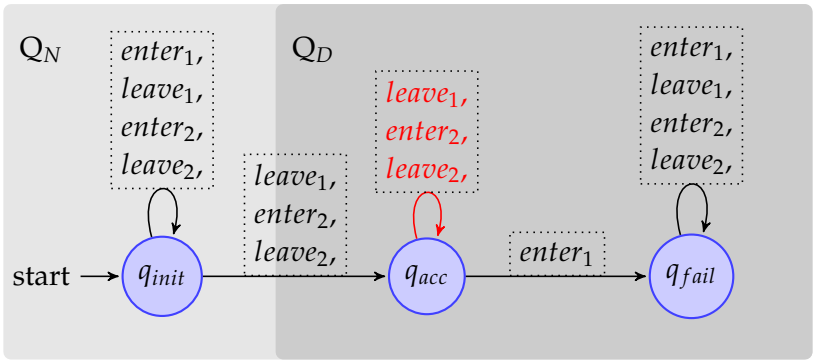


Figure 6.2.: Limit-deterministic Büchi automaton over the transitions of $\mathcal{P}_{SimpleMutex}$ with non-deterministic part Q_N and deterministic part Q_D.

6.2. Product of Protocol and LDBA

Definition 7. *Let $\mathcal{P} = (Q_\mathcal{P}, \mathcal{T}_\mathcal{P}, \Sigma_\mathcal{P}, \mathcal{I}_\mathcal{P}, \mathcal{O}_\mathcal{P})$ be a population protocol that is labeled by some function $label_\mathcal{P} : \mathcal{T}_\mathcal{P} \to \Psi$ over the alphabet Ψ. Let $B = (Q_B, \Psi, \delta_B, \mathcal{I}_B, F_B)$ be an LDBA with $Q_\mathcal{P} \cap Q_B = \varnothing$. The* product *of $\mathcal{P}$ and B is the protocol $\mathcal{P}_{\mathcal{P}\times B} = (Q_{\mathcal{P}\times B}, \mathcal{T}_{\mathcal{P}\times B}, \Sigma_{\mathcal{P}\times B}, \mathcal{I}_{\mathcal{P}\times B}, \mathcal{O}_{\mathcal{P}\times B})$ with*

$$
\begin{aligned}
Q_{\mathcal{P}\times B} &\stackrel{def}{=} Q_\mathcal{P} \cup Q_B \\
\mathcal{T}_{\mathcal{P}\times B} &\stackrel{def}{=} \left\{ \wr q \wr + \mathrm{pre}(t) \to \wr q' \wr + \mathrm{post}(t) \;\middle|\; t \in \mathcal{T}_\mathcal{P} \wedge q' \in \delta_B(q, label_\mathcal{P}(t)) \right\} \\
\Sigma_{\mathcal{P}\times B} &\stackrel{def}{=} \Sigma_\mathcal{P} \cup \mathcal{I}_B \\
\mathcal{I}_{\mathcal{P}\times B}(x) &\stackrel{def}{=} \begin{cases} x & \text{if } x \in \mathcal{I}_B \\ \mathcal{I}_\mathcal{P}(x) & \text{otherwise} \end{cases} && \text{for all } x \in \Sigma_{\mathcal{P}\times B} \\
\mathcal{O}_{\mathcal{P}\times B}(q) &\stackrel{def}{=} true && \text{for all } q \in Q_{\mathcal{P}\times B}
\end{aligned}
$$

Intuitively, the Büchi automaton monitors the transitions of the protocol. Whenever

a transition occurs, it changes the state of the Büchi automaton according to the label of the original transition.

6.3. Checking LTL

Theorem 6.3.1. *Let $\mathcal{P} = (Q_\mathcal{P}, \mathcal{T}_\mathcal{P}, \Sigma_\mathcal{P}, \mathcal{I}_\mathcal{P}, \mathcal{O}_\mathcal{P})$ be a population protocol that is labeled by some function $label_\mathcal{P} : \mathcal{T}_\mathcal{P} \to \Psi$ over the alphabet Ψ. Furthermore, let $B = (Q_B, \Psi, \delta_B, \mathcal{I}_B, F_B)$ be an LDBA with $Q_\mathcal{P} \cap Q_B = \varnothing$ and $L(B) = L(\neg\varphi)$ for some LTL predicate φ. Let $\mathcal{P}_{\mathcal{P}\times B} = (Q_{\mathcal{P}\times B}, \mathcal{T}_{\mathcal{P}\times B}, \Sigma_\mathcal{P}, \mathcal{I}_\mathcal{P}, \mathcal{O}_\mathcal{P})$ be the product of $\mathcal{P}$ and B. For $t \in \mathcal{T}_\mathcal{P}$ and $\sigma \in \mathcal{T}_\mathcal{P}^*$, we define the homomorphism h:*

$$h(t) \overset{def}{=} label_\mathcal{P}(t)$$
$$h(t\sigma) \overset{def}{=} h(t)h(\sigma)$$

We define the set of accepting transitions T_{acc}:

$$\mathcal{T}_{acc} \overset{def}{=} \left\{ \wr q \int + \mathrm{pre}(t) \to \wr q' \int + \mathrm{post}(t) \;\middle|\; t \in \mathcal{T}_\mathcal{P} \wedge (q, label_\mathcal{P}(t), q') \in F_B \right\}$$

We define the following Presburger predicates:

$$\varphi_{pre} \overset{def}{=} init_{\mathcal{P}\times B} \wedge \left(1 = \sum_{q \in \mathcal{I}_B} q \right)$$
$$\varphi_{post} = \textsc{Dis}(T_{acc})$$

where $init_{\mathcal{P}\times B}$ is the Presburger predicate describing all initial configurations of $\mathcal{P}_{\mathcal{P}\times B}$.

If there is a φ_{pre}-φ_{post}-stage-graph for $\mathcal{P}_{\mathcal{P}\times B}$, then for every fair execution $\pi = C_0 \xrightarrow{t_1} C_1 \xrightarrow{t_2}$ of $\mathcal{P}$ starting in an initial configuration C_0, it holds that $h(t_1 t_2 \cdots) \vDash \varphi$.

Proof. Assume for sake of contradiction that $h(t_1 t_2 \cdots) \nvDash \varphi$. Then we know that $h(t_1 t_2 \cdots) \vDash \neg\varphi$ and thus, $h(t_1 t_2 \cdots) \in L(B)$. Therefore, there exists an accepting path $\sigma = q_0 \xrightarrow{h(t_1)} q_1 \xrightarrow{h(t_2)} \cdots$ on B with $q_0 \in \mathcal{I}_B$.

We construct the execution $\pi' = C'_0 \xrightarrow{t'_1} C'_1 \xrightarrow{t'_2} \cdots$ for the protocol $\mathcal{P}_{\mathcal{P}\times B}$ such that for all $i \geq 0$:

$$C'_i \overset{\text{def}}{=} C_i + \wr q_i \int$$
$$t'_i \overset{\text{def}}{=} \wr q_i \int + \mathrm{pre}(t) \to \wr q_{i+1} \int + \mathrm{post}(t)$$

The execution π' is the combination of the run σ in B and the fair execution π in $\mathcal{P}$. Because σ is accepting, we know that π' uses the transitions $\mathcal{T}_{acc}$ infinitely often. Furthermore, $C'_0 \vDash \varphi_{pre}$.

We will now argue, that π' is a fair execution. Let $C'_j = C_j + \{\!|q_j|\!\}$ be a configuration of π' that is visited infinitely often and enables some transition $t' \in \mathcal{T}_{\mathcal{P}\times B}$. Thus, there is a transition $t \in \mathcal{T}_{\mathcal{P}}$ enabled in C_j. Because π is fair, t occurs infinitely often in π after C_j. As C'_j is visited infinitely often and σ is accepting, we know that q_j must be part of the deterministic component of B. This implies that t' is the only transition corresponding to t. Therefore, t' occurs infinitely in π' after C'_j. Hence, π' is fair.

As there is a φ_{pre}-φ_{post}-stage-graph for $\mathcal{P}_{\mathcal{P}\times B}$, we know using theorem 3.0.1 that $\mathbb{N}^{Q_{\mathcal{P}\times B}} \vDash \varphi_{pre} \Rightarrow FG\varphi_{post}$. Therefore, the transitions $\mathcal{T}_{acc}$ will eventually be dead when executing π'. In other words, π' uses the transitions $\mathcal{T}_{acc}$ only finitely often. This is a contradiction. □

6.4. Example

To verify the liveness property $GF(enter_1)$ for the population protocol $\mathcal{P}_{SimpleMutex}$ (see Figure 6.1) we first compute its product with the Büchi automaton shown in Figure 6.2. The resulting protocol has the states $Q = \{q_{init}, q_{acc}, q_{fail}, p_1, p_2, cs_1, cs_2, m, \neg m\}$ and the transitions of Table 6.1.

	pre	→	post	label	accepting?
t_1	q_{init}, p_1, m	→	$q_{init}, cs_1, \neg m$	$enter_1$	no
t_2	q_{acc}, p_1, m	→	$q_{fail}, cs_1, \neg m$	$enter_1$	no
t_3	q_{fail}, p_1, m	→	$q_{fail}, cs_1, \neg m$	$enter_1$	no
t_4	$q_{init}, cs_1, \neg m$	→	q_{init}, p_1, m	$leave_1$	no
t_5	$q_{init}, cs_1, \neg m$	→	q_{acc}, p_1, m	$leave_1$	no
t_6	$q_{acc}, cs_1, \neg m$	→	q_{acc}, p_1, m	$leave_1$	yes
t_7	$q_{fail}, cs_1, \neg m$	→	q_{fail}, p_1, m	$leave_1$	no
t_8	q_{init}, p_2, m	→	$q_{init}, cs_2, \neg m$	$enter_2$	no
t_9	q_{init}, p_2, m	→	$q_{acc}, cs_2, \neg m$	$enter_2$	no
t_{10}	q_{acc}, p_2, m	→	$q_{acc}, cs_2, \neg m$	$enter_2$	yes
t_{11}	q_{fail}, p_2, m	→	$q_{fail}, cs_2, \neg m$	$enter_2$	no
t_{12}	$q_{init}, cs_2, \neg m$	→	q_{init}, p_2, m	$leave_2$	no
t_{13}	$q_{init}, cs_2, \neg m$	→	q_{acc}, p_2, m	$leave_2$	no
t_{14}	$q_{acc}, cs_2, \neg m$	→	q_{acc}, p_2, m	$leave_2$	yes
t_{15}	$q_{fail}, cs_2, \neg m$	→	q_{fail}, p_2, m	$leave_2$	no

Table 6.1.: Transitions of the product protocol for $\mathcal{P}_{SimpleMutex}$ and the LDBA of Figure 6.2.

We define the precondition φ_{pre} such that the only possible initial configuration is

$\wr q_{init}, p_1, p_2, m \int$. The postcondition must make sure that all accepting transitions are disabled, i.e. $\varphi_{post} \stackrel{\text{def}}{=} \textsc{Dis}(\{t_6, t_{10}, t_{14}\})$.

The algorithm of Section 4 computes a φ_{pre}-φ_{post}-stage-graph for $\mathcal{P}_{\mathcal{P}\times B}$ with three stages (see Figure 6.3). Using Theorem 6.3.1, we know that any fair execution π of the protocol $\mathcal{P}_{SimpleMutex}$ uses the transition $enter_1$ infinitely often. Or in other words: The transition $enter_1$ will never be dead if we assume a fair scheduler.

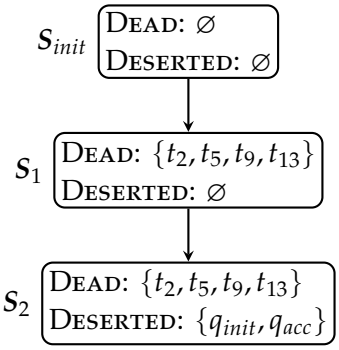


Figure 6.3.: Stage graph verifying the property $GF\ enter_1$ for the protocol $\mathcal{P}_{SimpleMutex}$.

7. Evaluation

We have implemented our approach as an extension of the tool *Peregrine* [18]. Peregrine is a tool for the analysis and parameterized verification of population protocols. With its graphical user interface, it allows the user to specify population protocols that then can be simulated both manually and automatically. Peregrine can also find executions of faulty protocols that do not converge to the correct output. By using the technique of [5], the tool can automatically verify if a given silent protocol computes a specified formula for all of the infinitely many initial configurations.

Our extension for Peregrine (called *Peregrine 2.0*) enables the verification of non-silent protocols and is embedded into the graphical interface. Additionally, we offer a command-line tool for more involved queries (e.g. arbitrary pre- and postconditions and liveness checks with LTL).

We implemented the main algorithm of Section 4 as a *PYTHON 3* backend that uses the SMT solver Z3 [19]. To convert an LTL formula into an LDBA we use the tool *OWL* [20]. The full implementation can be found in the Peregrine Git repository[1].

7.1. Benchmarks

We tested our implementation on multiple protocols from the literature. Some of those protocols are parametric, i.e. they are part of a parametric family of protocols.

In addition to the examples given in this work, we tested multiple majority protocols [6][7], Flock-of-Birds protocols [2][21], leader election protocols [22][23] as well as remainder and threshold protocols [2]. For Flock-of-Birds, remainder and threshold we also tested so-called *succinct* protocol families (see [6] and Appendix). The number of states in succinct protocol families only grows logarithmically in the parameters resulting in much smaller protocols.

All tests were performed on the same machine equipped with an Intel(R) Xeon(R) E5-2630 v4 CPU (@ 2.20GHz) and 256 GB of RAM. Please note that one can achieve comparable results with much less RAM (e.g. 8GB).

[1] `https://gitlab.lrz.de/ga97cer/peregrine/`

7.1.1. Parameters

In Section 4.4, we give multiple implementations for parts of the algorithm:

- PotInStage vs PotInStage$_{backwards}$
- EventuallyDead: *ranking* vs *layered* vs *combined*

Table 7.1 shows the impact of the different implementations on the success and the computation time of the algorithm for multiple population protocols.

In general, we can say that PotInStage$_{backwards}$ is a more accurate overapproximation when compared to PotInStage. However, this precision may cause longer computation times especially if the protocol is large.

In Section 4.4.1 we have shown that the implementations EventuallyDead$_{ranking}$ and EventuallyDead$_{layered}$ are incomparable. Thus, the combined approach for finding eventually dead transitions is superior to both of them. As the ranking function approach is needed in most cases, we do not compare the implementation that only uses EventuallyDead$_{layered}$. Although the combined approach solves additional constraints, it is sometimes faster than the ranking function approach (e.g. remainder protocol in Table 7.1). This can happen because finding more eventually dead transitions early on, can reduce the complexity of the problem and thus result in a smaller stage graph.

7.1.2. Scaling

Table 7.2 shows the computation time when computing stage graphs for increasingly large population protocols. The timeout is one hour.

In some cases, we can verify the correctness of population protocols with more than 70 states and over 2000 transitions. Our results for silent population protocols are comparable to the results given in [5], even though our implementation can handle a larger class of population protocols. Both implementations have similar computation times when run on the same protocol and thus handle protocols families up to comparably large parameters.

The highest split degree (i.e. the largest number of substages resulting from a split) is 2 for nearly all protocols. There are two exceptions:

- *Average and Conquer* Majority [7]:
  increases slowly for large protocols
- Threshold/Remainder [5] when using ranking transition approach:
  increases linearly with largest input parameter

EvDead	PotInStage	**Result**	**Time** (secs.)
$\mathcal{P}_{majority}$ (see Figure 2.1)			
ranking	default	failure	< 1
combined	default	success	< 1
ranking	backwards	failure	< 1
combined	backwards	success	< 1
$\mathcal{P}_{X\geq 3}$ (see Figure 2.2)			
ranking	default	failure	< 1
combined	default	failure	< 1
ranking	backwards	success	< 1
combined	backwards	success	< 1
Majority (Example 3 in [8])			
ranking	default	success	< 1
combined	default	success	< 1
ranking	backwards	success	< 1
combined	backwards	success	< 1
Remainder protocol [5]: $\sum_{1\leq i\leq 8} i * x_i \equiv_8 0$			
ranking	default	success	11.6
combined	default	success	3.4
ranking	backwards	success	13.0
combined	backwards	success	3.0
Threshold protocol [5]: $\sum_{-2\leq i\leq 2} i * x_i < 1$			
ranking	default	success	21.9
combined	default	success	7.4
ranking	backwards	success	29.6
combined	backwards	success	9.1

EvDead	PotInStage	**Result**	**Time** (secs.)
$\mathcal{P}_{majority}$ without tie-breaking rule t_{ab}, with $\varphi_{pre} = A \neq B$			
ranking	default	success	< 1
combined	default	success	< 1
ranking	backwards	success	< 1
combined	backwards	success	< 1
Broadcast [24]			
ranking	default	success	< 1
combined	default	success	< 1
ranking	backwards	success	< 1
combined	backwards	success	< 1
Average and Conquer Majority [7]: $m = 5$, $d = 3$			
ranking	default	success	15.1
combined	default	success	16.1
ranking	backwards	success	10.5
combined	backwards	success	16.8
succinct Remainder protocol (see A.4): $\sum_{1\leq i\leq 8} i * x_i \equiv_{15} 0$			
ranking	default	failure	4.1
combined	default	failure	4.3
ranking	backwards	failure	7.6
combined	backwards	success	4.9
succinct Threshold protocol [6]: $\sum_{-2\leq i\leq 2} i * x_i < 1$			
ranking	default	failure	4.4
combined	default	failure	23.6
ranking	backwards	failure	15.3
combined	backwards	success	27.9

Figure 7.1.: Results of experimental evaluation showing the used implementation for EventuallyDead and PotInStage together with the success of the algorithm and the execution time.

7.1.3. Verifying Correctness / Postcondition

Our algorithm was able to show correctness for all protocols we tested (compare Figure 7.2). Additionally, we successfully verified multiple leader election protocols (see Figure 7.3).

One notable exception is *Herman's* leader election algorithm [23]. The goal is to elect one leader from n processes that form a ring. In [23], it is proposed to model the algorithm using one bit per process such that a given agent is a leader if its bit value is equal to the bit value of his predecessor. The transitions allow any leader to invert its bit. Our algorithm is not able to show that there is eventually exactly one leader because no transitions become dead and no states become deserted.

However, we can verify Herman's leader election algorithm if we model leaders with tokens. In the modified version of the protocol (see A.5), a process is a leader if its bit value is 1.

parameters	$\lvert Q\rvert$	$\lvert\mathcal{T}\rvert$	Result	Time (secs.)
Flock-of-Birds [2]: $x \geq c$ with EventuallyDead$_{ranking}$ & PotInStage				
$c = 2$	3	6	success	< 1
$c = 5$	6	21	success	< 1
$c = 10$	11	66	success	1.8
$c = 20$	21	231	success	6.8
$c = 30$	31	496	success	16.9
$c = 40$	41	861	success	44.1
$c = 50$	51	1326	success	68.2
$c = 60$	61	1891	success	328
$c = 70$	71	2556	-	-
Flock-of-Birds [21]: $x \geq c$ with EventuallyDead$_{ranking}$ & PotInStage				
$c = 2$	3	3	success	< 1
$c = 5$	6	9	success	< 1
$c = 10$	11	19	success	< 1
$c = 15$	16	29	success	1.6
$c = 20$	21	39	success	33.7
$c = 25$	26	49	success	2056
$c = 30$	31	59	-	-
succinct Flock-of-Birds (see A.1): $x \geq c$ with EventuallyDead$_{ranking}$ & PotInStage				
$c = 2^5 - 1$	10	34	success	1.0
$c = 2^{10} - 1$	20	119	success	3.2
$c = 2^{15} - 1$	30	254	success	13.1
$c = 2^{20} - 1$	40	439	success	32.1
$c = 2^{25} - 1$	50	674	success	112
$c = 2^{30} - 1$	60	959	success	125
$c = 2^{35} - 1$	70	1294	success	334
$c = 2^{40} - 1$	80	1679	-	-
reversible succ. Flock-of-Birds (see A.2): $x \geq c$ with EventuallyDead$_{ranking}$ & PotInStage$_{backwards}$				
$c = 31$	10	25	success	2.3
$c = 63$	12	31	success	39.8
$c = 127$	14	37	-	-
k-way reversible succ. Flock-of-Birds (see A.3): $x \geq c$ with EventuallyDead$_{ranking}$ & PotInStage$_{backwards}$				
$c = 31$	7	15	success	2.3
$c = 63$	8	16	success	39.8
$c = 127$	9	21	-	-
Remainder [5]: $\sum_{1 \leq i < k} i \cdot x_i \equiv_k 0$ with EventuallyDead$_{combined}$ & PotInStage				
$k = 5$	7	25	success	1.5
$k = 10$	12	75	success	4.5
$k = 15$	17	150	success	20.8
$k = 20$	22	250	success	565
$k = 25$	27	375	-	-

parameters	$\lvert Q\rvert$	$\lvert\mathcal{T}\rvert$	Result	Time (secs.)
succinct Remainder (see A.4): $\sum_{1 \leq i < k} i \cdot x_i \equiv_k 0$ with EventuallyDead$_{combined}$ & PotInStage$_{backwards}$				
$k = 15$	14	33	success	4.2
$k = 31$	15	37	success	8.5
$k = 63$	16	41	success	75.0
$k = 127$	27	375	-	-
Threshold [5]: $\sum_{-2 \leq i \leq 2} i \cdot x_i < c$ with EventuallyDead$_{combined}$ & PotInStage$_{backwards}$				
$c = -2$	28	301	success	16.6
$c = -1$	20	155	success	7.2
$c = 0$	20	155	success	7.6
$c = 1$	20	155	success	7.5
$c = 2$	28	301	success	15.6
$c = 3$	36	495	success	32.0
$c = 4$	44	737	success	60.0
$c = 5$	52	1027	success	100
$c = 6$	60	1365	success	174
$c = 7$	68	1751	success	309
$c = 8$	76	2185	success	445
$c = 9$	84	2667	-	-
succinct Threshold [6]: $\sum_{-2 \leq i \leq 2} i \cdot x_i < c$ with EventuallyDead$_{combined}$ & PotInStage$_{backwards}$				
$c = 3$	12	30	success	4.9
$c = 7$	14	39	success	15.6
$c = 15$	16	48	success	32.0
$c = 31$	18	57	success	60.0
$c = 63$	20	66	success	100
$c = 127$	22	75	success	174
$c = 255$	24	84	success	309
$c = 511$	26	93	success	445
$c = 1023$	28	102	-	-
Average and Conquer Majority [7] with EventuallyDead$_{combined}$ & PotInStage				
$m = 3, d = 1$	6	21	success	1.8
$m = 3, d = 2$	8	36	success	4.1
$m = 5, d = 1$	8	36	success	3.1
$m = 5, d = 4$	14	105	success	32.9
$m = 7, d = 1$	10	55	success	5.6
$m = 7, d = 6$	20	210	success	171
$m = 9, d = 1$	12	78	success	8.7
$m = 9, d = 8$	26	351	success	608
$m = 11, d = 1$	14	105	success	11.9
$m = 11, d = 6$	24	300	success	229
$m = 11, d = 7$	26	351	success	397
$m = 11, d = 8$	28	406	success	700
$m = 11, d = 9$	30	465	success	1177
$m = 11, d = 10$	32	528	success	1866
$m = 13, d = 1$	16	136	success	18.9
$m = 13, d = 12$	38	741	-	-

Figure 7.2.: Results of experimental evaluation for protocol families with increasingly large parameters. $\lvert Q\rvert$ and $\lvert\mathcal{T}\rvert$ correspond to the number of states and transitions in the protocol. The timeout is one hour.

parameters	$\|Q\|$	$\|\mathcal{T}\|$	**Result**	**Time** (secs.)
$\mathcal{P}_{leader}$ (see Figure 5.1) with EventuallyDead$_{ranking}$ & PotInStage				
-	2	1	success	< 1
Israeli-Jalfon (USSME) [22] with EventuallyDead$_{ranking}$ & PotInStage*backwards*				
n = 20	40	80	success	5.4
n = 40	80	160	success	24.1
n = 60	120	240	success	1161
n = 70	140	280	success	2537
n = 80	160	320	-	-
Herman [23] with EventuallyDead$_{combined}$ & PotInStage*backwards*				
n = 111	222	222	failure	258
n = 231	462	462	failure	2141
Herman modified (see A.5) with EventuallyDead$_{combined}$ & PotInStage*backwards*				
n = 11	22	22	success	1.2
n = 31	62	62	success	28.8
n = 51	102	102	success	233
n = 71	142	142	success	741
n = 91	182	182	success	2785
n = 111	222	222	-	-

Figure 7.3.: Results of experimental evaluation when verifying multiple leader election algorithms. $|Q|$ and $|\mathcal{T}|$ correspond to the number of states and transitions in the protocol. The timeout is one hour.

7.1.4. Speed Bounds

We evaluated the speed bounds that are computed as described in Section 5. The results can be seen in Table 7.4. When compared to the results given in [8], we see that our approach can compute the same speed bounds while verifying correctness. Furthermore, it is considerably faster and can consequently compute speed bounds for larger population protocols.

Protocol	PotInStage	**Speed**	**Time** (secs.)
$\mathcal{P}_{majority}$ (see Figure 2.1)	backwards	$e^{O(n \log n)}$	< 1
$\mathcal{P}_{majority}$ without tie-breaking rule t_{ab}, with $\varphi_{pre} = A \neq B$	default	$O(n^2 \log n)$	< 1
$\mathcal{P}_{X \geq 3}$ (see Figure 2.2)	default	$O(n^c)$	< 1
Broadcast [24]	default	$O(n^2 \log n)$	< 1
Majority (Example 3 in [8])	default	$O(n^2 \log n)$	1.3
Leader election (see Figure 5.1)	default	$O(n^2 \log n)$	< 1
Israeli-Jalfon (USSME) [22]: $n = 40$	backwards	$O(n^c)$	124
Flock-of-Birds [2]: $x \geq 45$	default	$O(n^3)$	307
Flock-of-Birds [21]: $x \geq 26$	default	$O(n^3)$	24.9
succinct Flock-of-Birds (see A.1): $x \geq 511$	default	$O(n^3)$	2.5
reversible succinct Flock-of-Birds (see A.2): $x \geq 63$	backwards	$O(n^c)$	40.9
Remainder [5]: $\sum_{1 \leq i < 4} i \cdot x_i \equiv_4 0$	default	$O(n^2 \log n)$	2.8
Threshold [5]: $\sum_{-2 \leq i \leq 2} i \cdot x_i < 2$	default	$O(n^3)$	2.8
Average and Conquer Majority [7]: $m = 3, d = 1$	default	$O(n^2 \log n)$	1.7
Average and Conquer Majority [7]: $m = 3, d = 2$	default	$O(n^2 \log n)$	5.2
Average and Conquer Majority [7]: $m = 5, d = 1$	default	$O(n^2 \log n)$	3.9
Average and Conquer Majority [7]: $m = 5, d = 2$	default	$O(n^3)$	10.4
Average and Conquer Majority [7]: $m = 5, d = 3$	default	$O(n^3)$	14.4
Average and Conquer Majority [7]: $m = 5, d = 4$	default	$O(n^3)$	20.5
Average and Conquer Majority [7]: $m = 7, d = 1$	default	$O(n^2 \log n)$	8.3
Average and Conquer Majority [7]: $m = 7, d = 6$	default	$O(n^3)$	60.6
Average and Conquer Majority [7]: $m = 9, d = 1$	default	$O(n^3)$	12.9
Average and Conquer Majority [7]: $m = 9, d = 8$	default	$O(n^3)$	201
Average and Conquer Majority [7]: $m = 11, d = 1$	default	$O(n^3)$	25.3
Average and Conquer Majority [7]: $m = 11, d = 10$	default	$O(n^3)$	550

Figure 7.4.: Calculated speed bounds while successfully verifying multiple population protocols.

7.1.5. More systems and LTL

We used our algorithm to successfully check liveness properties for multiple mutex algorithms (see Table 7.5). The timeout is one hour. We modeled each algorithm as closely as possible by a population protocol. Then we use the approach of Section 6 to show liveness of a process. The specific implementations of the mutex algorithms as population protocols can be found in the Git repository for Peregrine[2].

[2] `https://gitlab.lrz.de/ga97cer/peregrine/`

processes	$\|Q\|$	$\|\mathcal{T}\|$	LTL	Time (secs.)
$\mathcal{P}_{SimpleMutex}$ (6.1)				
2	8	15	$GF\ enter_1$	< 1
100	204	799	$GF\ enter_1$	62.3
200	404	1599	$GF\ enter_1$	298
300	604	2399	$GF\ enter_1$	844
400	804	3199	$GF\ enter_1$	2049
500	1004	3999	$GF\ enter_1$	-
Array [25]				
2	15	47	$GF\ enter_1$	< 1
8	51	191	$GF\ enter_1$	39.0
9	57	215	$GF\ enter_1$	142
10	63	239	$GF\ enter_1$	551
11	69	263	$GF\ enter_1$	2284
12	75	287	$GF\ enter_1$	-
Burns [26]				
2	11	43	$GF\ enter_1$	1.3
3	15	83	$GF\ enter_1$	2.9
4	19	139	$GF\ enter_1$	8.2
5	23	199	$GF\ enter_1$	67.1
6	27	275	$GF\ enter_1$	1074
7	31	287	$GF\ enter_1$	-
Peterson [27]				
2	13	46	$GF\ enter_1$	1.6

processes	$\|Q\|$	$\|\mathcal{T}\|$	LTL	Time (secs.)
Dijkstra [28]				
2	17	124	$GF\ enter_1$	4.1
3	24	368	$GF\ enter_1$	37.8
4	31	1280	$GF\ enter_1$	3221
5	38	5344	$GF\ enter_1$	-
Szymanski [29]				
2	17	155	$GF\ enter_1$	5.6
3	24	811	$GF\ enter_1$	37.8
4	31	4795	$GF\ enter_1$	-
Lehmann Rabin [30]				
2	19	71	$GF\ eat_1$	4.3
7	59	251	$GF\ eat_1$	355
8	67	287	$GF\ eat_1$	691
9	75	323	$GF\ eat_1$	1499
10	83	359	$GF\ eat_1$	3141
11	91	395	$GF\ eat_1$	-
Kanban [31]				
2	19	71	$GF\ produce$	4.3
7	59	251	$GF\ produce$	355
8	67	287	$GF\ produce$	691
9	75	323	$GF\ produce$	1499
10	83	359	$GF\ produce$	3141
11	91	395	$GF\ produce$	-

Figure 7.5.: Successfully verifying liveness of some process for multiple mutex algorithms using $\text{EventuallyDead}_{ranking}$ and $\text{PotInStage}_{backwards}$. $|Q|$ and $|\mathcal{T}|$ correspond to the number of states and transitions in the product protocol. The timeout is one hour.

8. Conclusion

We introduced stage graphs, a structure describing all possible fair executions of a population protocol. We have shown that stage graphs can be used as certificates verifying properties of population protocols such as correctness.

Furthermore, we presented an algorithm that can compute a stage graph that verifies a given property. We extended the algorithm to additionally compute an upper bound on the expected number of transitions until termination. Together with a product construction for Limit-Deterministic-Büchi-Automata and population protocols, we can even verify LTL formulas.

Our algorithm can prove the correctness of most protocols in the literature and computes good speed bounds. On top of that, it can verify liveness for multiple mutex algorithms if the number of processes is small.

An interesting question is whether our approach can be used with more expressive models like Petri nets with inhibitor arcs or population protocols with broadcast transitions.

Another venue for future research is the compatibility of stage graphs with other fairness assumptions.

A. More Population Protocols

In this chapter we give the definitions for protocols of the form $\mathcal{P} = (Q, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ that compute some predicate φ.

Let $n \in \mathbb{N}_{>0}$. We define $\text{bits}(n)$ as the set of indices of the bits occurring in the binary representation of n, e.g. $\text{bits}(13) = \{0, 2, 3\}$ since $13 = 1101_2$. The size of n, denoted $\text{size}(n)$, is the number of bits required to represent n in binary. Note that $|\text{bits}(n)| \leq \text{size}(n) = \lfloor \log_2 n \rfloor + 1$.

A.1. Protocol: succinct Flock of Birds

A succinct (i.e. small) protocol family for the Flock of Birds predicate $X \geq c$ for some positive parameter c. The protocol family is silent. The number of states (i.e. the size of the protocol) is $O(\log c)$. Let $k \stackrel{\text{def}}{=} \lfloor \log_2(c) \rfloor$.

$$Q \stackrel{\text{def}}{=} \{0\} \cup \left\{ 2^i \,\middle|\, 0 \leq i \leq k \right\} \cup \left\{ \sum_{\substack{j=i \\ j \in \text{bits}(c)}}^{k} 2^j \,\middle|\, 0 \leq i \leq k \right\}$$

$$\mathcal{T}(p,q) \stackrel{\text{def}}{=} \begin{cases} c, c & \text{if } p + q \geq c \\ (p+q), 0 & \text{if } (p+q) \in Q \\ p, q & \text{otherwise} \end{cases}$$

$$\Sigma \stackrel{\text{def}}{=} \{X\}$$

$$\mathcal{I}(X) \stackrel{\text{def}}{=} 1$$

$$\mathcal{O}(q) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } q = c \\ 0 & \text{otherwise} \end{cases}$$

$$\varphi \stackrel{\text{def}}{=} X \geq c$$

A.2. Protocol: succinct reversible Flock of Birds

Same as Protocol A.1 but for each transition of the form $p, q \to (p+q), 0$ we additionally add the reverse transition $(p+q), 0 \to p, q$. The protocol family is non-silent.

A.3. Protocol: k-way succinct reversible Flock of Birds

A succinct (i.e. small) protocol family for the Flock of Birds predicate $X \geq c$ for some positive parameter c. Compared to the 2-way protocols A.1 and A.2 this protocol has transitions involving more agents. The protocol family is non-silent because of reverse transitions. The number of states (i.e. the size of the protocol) is $O(\log c)$. Let $k \stackrel{\text{def}}{=} \lfloor \log_2(c) \rfloor$.

$$
\begin{aligned}
\mathrm{Q} &\stackrel{\text{def}}{=} \{0, c\} \cup \left\{ 2^i \,\middle|\, 0 \leq i \leq k \right\} \\
\Sigma &\stackrel{\text{def}}{=} \{X\} \\
\mathcal{I}(X) &\stackrel{\text{def}}{=} 1 \\
\mathcal{O}(q) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } q = c \\ 0 & \text{otherwise} \end{cases} \\
\varphi &\stackrel{\text{def}}{=} X \geq c
\end{aligned}
$$

and the transitions $\mathcal{T}$ are:

$$
\begin{aligned}
2^i, 2^i &\to 2^{i+1}, 0 && \text{for every } 0 \leq i < k \\
2^{i+1}, 0 &\to 2^i, 2^i && \text{for every } 0 \leq i < k \\
\wr 2^i \mid i \in \mathrm{bits}(c) \wr &\to \underbrace{c, c, \cdots, c}_{|\mathrm{bits}(c)| \text{ times}} \\
c, q &\to c, c && \text{for every } q \in \mathrm{Q}
\end{aligned}
$$

A.4. Protocol: succinct Remainder

Let φ be a remainder predicate of the form $\sum_{1 \leq i \leq k} a_i x_i \equiv_m c$ with $m \geq 2$.

We will now describe the corresponding protocol $P_{rem} \stackrel{\text{def}}{=} (\mathrm{Q}, \mathcal{T}, \Sigma, \mathcal{I}, \mathcal{O})$ with size $O(k + \log m)$ and a multiset L of leaders of size $O(\log m)$. Those leader agents are present independent of the input and can be modeled as a precondition. The protocol is non-silent.

Let $n \stackrel{\text{def}}{=} \text{size}(m) - 1$. The states are $Q \stackrel{\text{def}}{=} N + R + X$ with

$$N \stackrel{\text{def}}{=} \left\{2^i \,\middle|\, 0 \le i \le n\right\} \quad \text{and} \quad R \stackrel{\text{def}}{=} \{0, 0^-\} \quad \text{and} \quad X \stackrel{\text{def}}{=} \{x_i \,|\, 1 \le i \le k\}.$$

The states in N represent values for powers of 2 and X contains the input states for all variables. The states in R are "reservoir states" with value 0, but they have different outputs. The states 0 has output *true* whereas the state 0^- has output *false*. Intuitively, the value 0 has the default output *true*, but it can be "persuaded" to have the output *false*. The alphabet is $\Sigma \stackrel{\text{def}}{=} X$ and the input function $\mathcal{I}$ maps x_i to the state x_i.

We will define the function $\text{rep} : \mathbb{N} \to \mathbb{N}^Q$ as follows:

$$\text{rep}(a) \stackrel{\text{def}}{=} \begin{cases} \wr 2^i \,|\, i \in \text{bits}(a) \wr & \text{if } a > 0 \\ \wr 0 \wr & \text{if } a = 0 \end{cases}$$

The leaders are $L \stackrel{\text{def}}{=} \text{rep}((m-c) \;\%\; m) + \underbrace{\wr 0, \cdots, 0 \wr}_{2n+2 \text{ times}}$. They have two purposes: The first is to make sure that the predicate is true if the overall value of all agents is a multiple of m. The second task is that we need a reservoir of neutral agents in state 0. These reservoir agents are needed, so that all k-way transitions from input states to value states can occur. For each $1 \le i \le k$ and $r \in R$, we add the transition:

$$add_{x_i,r} : x_i, \underbrace{r, r, ..., r}_{|\text{rep}(a_i)|-1 \text{ times}} \to \text{rep}(a_i)$$

We will allow two interacting agents to change into an equivalent representation of their combined values. For each $0 \le i < n$ and $r \in R$ we add the transitions:

$$up_i : 2^i, 2^i \to 2^{i+1}, 0^-$$
$$down_{i,r} : 2^{i+1}, r \to 2^i, 2^i$$

For example, agents with values 4 and 4 can interact and result in values 8 and 0. Also, the reverse of this interaction is allowed.

Now we add another transition that models "mod m" by reducing the overall value by m:

$$mod : \text{rep}(m) \to \underbrace{0, 0, ..., 0}_{|\text{rep}(m)| \text{ times}}$$

Finally, we need transitions that can persuade the agents with value 0. For each $0 \le i \le n$, we add the transitions:

$$signal : 0, 0^- \to 0, 0$$
$$signal_i : 2^i, 0 \to 2^i, 0^-$$

The reason for *signal* is that if there are no more agents with positive values left, all agents should output *true*. But whenever there is an agent with a positive value, the transitions $signal_i$ set the output to *false*.

A.5. Protocol: Herman (modified)

This protocol models Herman's leader election protocol [23] with tokens instead of bits. Intuitively, a process is a leader if it has a token. For an odd number n of processes, the protocol is defined as:

$$
\begin{aligned}
&\mathrm{Q} \stackrel{\text{def}}{=} \{leader_i \mid 1 \le i \le n\} \cup \{\neg leader_i \mid 1 \le i \le n\} \\
&\mathcal{T} \stackrel{\text{def}}{=} \left\{leader_i, leader_{(i+1)\%n} \to \neg leader_i, \neg leader_{(i+1)\%n} \;\middle|\; 1 \le i \le n\right\} \\
&\qquad \cup \left\{leader_i, \neg leader_{(i+1)\%n} \to \neg leader_i, leader_{(i+1)\%n} \;\middle|\; 1 \le i \le n\right\} \\
&\Sigma \stackrel{\text{def}}{=} \mathrm{Q} \\
&\mathcal{I}(q) \stackrel{\text{def}}{=} q \qquad \text{for every } q \in \mathrm{Q} \\
&\mathcal{O}(q) \stackrel{\text{def}}{=} true \qquad \text{for every } q \in \mathrm{Q}
\end{aligned}
$$

The pre- and postcondition for verifying the leader election are:

$$
\begin{aligned}
\varphi_{pre} &\stackrel{\text{def}}{=} \bigwedge_{i=1}^{n} (leader_i + \neg leader_i = 1) \\
&\qquad \cup \left(\sum_{i=1}^{n} leader_i\right) \equiv_2 1 \\
\varphi_{post} &\stackrel{\text{def}}{=} \left(\sum_{i=1}^{n} leader_i\right) = 1
\end{aligned}
$$

List of Figures

Bibliography

[1] D. Angluin, J. Aspnes, M. Chan, M. J. Fischer, H. Jiang, and R. Peralta, "Stably computable properties of network graphs", in *International Conference on Distributed Computing in Sensor Systems*, Springer, 2005, pp. 63–74.

[2] D. Angluin, J. Aspnes, Z. Diamadi, M. J. Fischer, and R. Peralta, "Computation in networks of passively mobile finite-state sensors", *Distributed Computing*, pp. 235–253, Mar. 2006.

[3] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar, "Verification of population protocols", *Acta Informatica*, vol. 54, no. 2, pp. 191–215, Mar. 2017, ISSN: 1432-0525. DOI: 10.1007/s00236-016-0272-3.

[4] W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki, "The reachability problem for petri nets is not elementary", in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, ACM, 2019, pp. 24–33.

[5] M. Blondin, J. Esparza, S. Jaax, and P. J. Meyer, "Towards efficient verification of population protocols", in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, ACM, 2017, pp. 423–430.

[6] M. Blondin, J. Esparza, and S. Jaax, "Large flocks of small birds: On the minimal size of population protocols", in *35th Symposium on Theoretical Aspects of Computer Science, STACS 2018, February 28 to March 3, 2018, Caen, France*, 2018, 16:1–16:14. DOI: 10.4230/LIPIcs.STACS.2018.16. [Online]. Available: https://doi.org/10.4230/LIPIcs.STACS.2018.16.

[7] D. Alistarh, R. Gelashvili, and M. Vojnović, "Fast and exact majority in population protocols", in *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, ACM, 2015, pp. 47–56.

[8] M. Blondin, J. Esparza, and A. Kucera, "Automatic analysis of expected termination time for population protocols", in *Proc. 29th International Conference on Concurrency Theory (CONCUR)*, 2018, 33:1–33:16. DOI: 10.4230/LIPIcs.CONCUR.2018.33.

[9] J. Desel and J. Esparza, *Free choice Petri nets*. Cambridge university press, 2005, vol. 40.

[10] P. H. Starke, *Analyse von Petri-netz-modellen*. Springer, 1990, vol. 6.

[11] P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay, "General decidability theorems for infinite-state systems", in *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, IEEE, 1996, pp. 313–321.

[12] A. Finkel and P. Schnoebelen, "Well-structured transition systems everywhere!", *Theoretical Computer Science*, vol. 256, no. 1-2, pp. 63–92, 2001.

[13] L. Bozzelli and P. Ganty, "Complexity analysis of the backward coverability algorithm for vass", in *Reachability Problems*, G. Delzanno and I. Potapov, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 96–109, ISBN: 978-3-642-24288-5.

[14] J. Esparza, P. Ganty, J. Leroux, and R. Majumdar, "Model checking population protocols", in *36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India*, 2016, 27:1–27:14. DOI: 10.4230/LIPIcs.FSTTCS.2016.27.

[15] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification", in *Proceedings of the First Symposium on Logic in Computer Science*, IEEE Computer Society, 1986, pp. 322–331.

[16] J. Esparza, J. J. Kretinsky, and S. Sickert, "One theorem to rule them all: A unified translation of LTL into ω-automata", in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, 2018, pp. 384–393. DOI: 10.1145/3209108.3209161. [Online]. Available: https://doi.org/10.1145/3209108.3209161.

[17] C. Courcoubetis and M. Yannakakis, "The complexity of probabilistic verification", *Journal of the ACM (JACM)*, vol. 42, no. 4, pp. 857–907, 1995.

[18] M. Blondin, J. Esparza, and S. Jaax, "Peregrine: A tool for the analysis of population protocols", in *Computer Aided Verification*, H. Chockler and G. Weissenbacher, Eds., Cham: Springer International Publishing, 2018, pp. 604–611, ISBN: 978-3-319-96145-3.

[19] L. de Moura and N. Bjørner, "Z3: An efficient smt solver", in *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and J. Rehof, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 337–340, ISBN: 978-3-540-78800-3. DOI: 10.1007/978-3-540-78800-3_24.

[20] J. Kretinsky, T. Meggendorfer, and S. Sickert, "Owl: A library for ω-words, automata, and LTL", in *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, 2018, pp. 543–550.

[21] J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu, "Guidelines for the verification of population protocols", in *2011 31st International Conference on Distributed Computing Systems*, IEEE, 2011, pp. 215–224.

[22] A. Israeli and M. Jalfon, "Token management schemes and random walks yield self-stabilizing mutual exclusion", in *Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, ACM, 1990, pp. 119–131.

[23] T. Herman, "Probabilistic self-stabilization", *Information Processing Letters*, vol. 35, no. 2, pp. 63–67, 1990.

[24] J. Clément, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu, "Guidelines for the verification of population protocols", in *2011 31st International Conference on Distributed Computing Systems*, IEEE, 2011, pp. 215–224.

[25] L. Fribourg and H. Olsén, "Reachability sets of parameterized rings as regular languages", *Electronic Notes in Theoretical Computer Science*, vol. 9, p. 40, 1997.

[26] M. Nilsson, "Regular model checking", PhD thesis, Uppsala University, 2000.

[27] W. Fokkink, *Distributed algorithms: an intuitive approach*. MIT Press, 2013.

[28] P. A. Abdulla, G. Delzanno, N. B. Henda, and A. Rezine, "Regular model checking without transducers (on efficient verification of parameterized systems)", in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2007, pp. 721–736.

[29] B. K. Szymanski, "A simple solution to lamport's concurrent programming problem with linear wait", in *Proceedings of the 2nd international conference on Supercomputing*, ACM, 1988, pp. 621–626.

[30] D. Lehmann and M. O. Rabin, "On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem", in *Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, ACM, 1981, pp. 133–138.

[31] G. Ciardo and M. Tilgner, "On the use of kronecker operators for the solution of generalized stochastic petri nets", 1996.