Verifying qualitative liveness properties of replicated systems with stochastic scheduling

Javier Esparza
Joint work with Michael Blondin, Martin Helfrich, Antonin Kucera, and Philipp J. Meyer
• **Replicated systems:** formal model to describe “swarms” of identical finite-state agents
• **Replicated systems:** formal model to describe “swarms” of identical finite-state agents

• Very simple: set of states and set of multiset rewriting rules $M \rightarrow M'$ where $|M| = |M'|$. 

Infinite sets of initial configurations allowed

Stochastic scheduling: agents to move next are chosen stochastically

Can model (abstractions of) multithreaded programs, population protocols and other distributed consensus algorithms, and chemical reaction networks
• **Replicated systems**: formal model to describe “swarms” of identical finite-state agents
  
  • Very simple: set of states and set of multiset rewriting rules $M \rightarrow M'$ where $|M| = |M'|$.
  
  • Infinite sets of initial configurations allowed
• **Replicated systems:** formal model to describe “swarms” of identical finite-state agents

• Very simple: set of states and set of multiset rewriting rules $M \rightarrow M'$ where $|M| = |M'|$.

• Infinite sets of initial configurations allowed

• **Stochastic scheduling:** agents to move next are chosen stochastically
Overview

- **Replicated systems:** formal model to describe “swarms” of identical finite-state agents
- Very simple: set of states and set of multiset rewriting rules $M \rightarrow M'$ where $|M| = |M'|$.
- Infinite sets of initial configurations allowed
- **Stochastic scheduling:** agents to move next are chosen stochastically
- Can model (abstractions of) multithreaded programs, population protocols and other distributed consensus algorithms, and chemical reaction networks
• indistinguishable mobile agents with very few resources
• agents change states via random pairwise interactions
• each state is labeled with an opinion true/false
• protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
Population protocols

• indistinguishable mobile agents with very few resources
• agents change states via random pairwise interactions
• each state is labeled with an opinion true/false
• protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
• indistinguishable mobile agents with very few resources
• agents change states via random **pairwise interactions**
• each state is labeled with an opinion true/false
• protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
• indistinguishable mobile agents with very few resources
• agents change states via random pairwise interactions
• each state is labeled with an opinion true/false
• protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion
Population protocols

- indistinguishable mobile agents with very few resources
- agents change states via random pairwise interactions
- each state is labeled with an opinion true/false
- protocol computes by stabilizing agents to common opinion

![Image of birds with sunglasses and thumbs down]
Example: majority protocol

At least as many blue birds than red birds?
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many **blue birds** than **red birds**?

**Protocol:**

- Two large birds of different colors become small and blue
- Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
At least as many **blue birds** than **red birds**?

**Protocol:**

- Two large birds of different colors become small and blue.
- Large birds convert small birds to their color.
At least as many **blue birds** than **red birds**?

**Protocol:**

- Two large birds of different colors become small and blue
- Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

• Two large birds of different colors become small and blue

• Large birds convert small birds to their color
Example: majority protocol

At least as many blue birds than red birds?

Protocol:

- Two large birds of different colors become small and blue
- Large birds convert small birds to their color
- **To break ties:** small blue birds convert small red birds
Example: threshold protocol

Are there at least 4 sick birds?
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\)
  if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\)
  if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \( \{0, 1, 2, 3, 4\} \)

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\)
  if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\)
  if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

• Each bird is in a state of \{0, 1, 2, 3, 4\}

• Sick birds initially in state 1 and healthy birds in state 0

• \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)

• \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \(\{0, 1, 2, 3, 4\}\)
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Are there at least 4 sick birds?

**Protocol:**

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \{0, 1, 2, 3, 4\}
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\) if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\) if \(m + n \geq 4\)
Example: threshold protocol

Are there at least 4 sick birds?

Protocol:

- Each bird is in a state of \( \{0, 1, 2, 3, 4\} \)
- Sick birds initially in state 1 and healthy birds in state 0
- \((m, n) \mapsto (m + n, 0)\)
  - if \(m + n < 4\)
- \((m, n) \mapsto (4, 4)\)
  - if \(m + n \geq 4\)
Replicated systems: formal model

- States: finite set $Q$
- Transitions: $T \subseteq \bigcup_{k \geq 2} Q^{(k)} \times Q^{(k)}$
Replicated systems: formal model

- States: finite set $Q$
- Transitions: $T \subseteq \bigcup_{k \geq 2} Q^{(k)} \times Q^{(k)}$
Replicated systems: formal model

• Configurations: $Q \rightarrow \mathbb{N}$

• Transitions induce step relation $C \rightarrow C'$ between configurations
Replicated systems: formal model

- Configurations: $Q \rightarrow \mathbb{N}$
- Transitions induce step relation $C \rightarrow C'$ between configurations
Replicated systems: computations

Reachability graph for \((3, 2, 0, 0)\):
Underlying Markov chain:
(pairs of agents are picked uniformly at random)
**Run:** infinite path from initial configuration
Qualitative model checking:

- LTL with Presburger formulas as atomic propositions
- Problem: decide if the runs satisfying the property have probability 1
- Unsurprisingly: not even semi-decidable
Qualitative model checking:

- LTL with Presburger formulas as atomic propositions
- Problem: decide if the runs satisfying the property have probability 1
- Unsurprisingly: not even semi-decidable

Decidable fragment: stable termination

\[ \text{Pre} \rightarrow F \left( \bigvee_{i=1}^{k} G \text{ Post}_i \right) \]
Correctness of population protocols:

- **Run converges to** \( b \) if from some moment on all agents only visit states of opinion \( b \)
- **Protocol computes** \( \varphi : \text{InitC} \rightarrow \{0, 1\} \):
  for every \( C \in \text{InitC} \), the runs starting at \( C \) reach **stable consensus** \( \varphi(C) \) with probability 1
Correctness of population protocols:

- Run **converges to** $b$ if from some moment on all agents only visit states of opinion $b$.

- **Protocol computes** $\varphi : \text{InitC} \rightarrow \{0, 1\}$:
  for every $C \in \text{InitC}$, the runs starting at $C$ reach **stable consensus** $\varphi(C)$ with probability 1.

Given a protocol and a predicate, the protocol computes the predicate iff the formula

$$\text{Init}_0 \rightarrow F \left( \bigvee_{i=1}^k G \text{Cons}_0 \right) \land \text{Init}_1 \rightarrow F \left( \bigvee_{i=1}^k G \text{Cons}_1 \right)$$

holds with probability 1.
Protocols can become complex, even for $\mathbf{B \geq R}$:

### Fast and Exact Majority in Population Protocols

Dan Alistarh  
Microsoft Research  

Rati Gelashvili*  
MIT  

Milan Vojnović  
Microsoft Research

1. \( \text{weight}(x) = \begin{cases} |x| & \text{if } x \in \text{StrongStates or } x \in \text{WeakStates}; \\ 1 & \text{if } x \in \text{IntermediateStates}. \end{cases} \)

2. \( \text{sgn}(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1d, \ldots, 11, 3, 5, \ldots, m\}; \\ -1 & \text{otherwise}. \end{cases} \)

3. \( \text{value}(x) = \text{sgn}(x) \cdot \text{weight}(x) \)

4. \( \phi(x) = -1_i \text{ if } x = -1; 1_i \text{ if } x = 1; x, \text{ otherwise} \)

5. \( R_i(k) = \phi(k \text{ if } k \text{ odd integer}, k - 1 \text{ if } k \text{ even}) \)

6. \( R_i(k) = \phi(k \text{ if } k \text{ odd integer}, k + 1 \text{ if } k \text{ even}) \)

7. \( \text{Shift-to-Zero}(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\ 1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\ x & \text{otherwise}. \end{cases} \)

8. \( \text{Sign-to-Zero}(x) = \begin{cases} +0 & \text{if } \text{sgn}(x) > 0 \\ -0 & \text{otherwise}. \end{cases} \)

9. \( \text{procedure } \text{update}(x, y) \)

10. if \((\text{weight}(x) > 0 \text{ and } \text{weight}(y) > 1) \text{ or } (\text{weight}(y) > 0 \text{ and } \text{weight}(x) > 1)\) then

11. \( x' \leftarrow R_i\left(\frac{\text{value}(x) + \text{value}(y)}{2}\right) \) and \( y' \leftarrow R_i\left(\frac{\text{value}(x) + \text{value}(y)}{2}\right) \)

12. else if \((\text{weight}(x) \cdot \text{weight}(y) = 0 \text{ and } \text{value}(x) + \text{value}(y) > 0)\) then

13. if \((\text{weight}(x) \neq 0)\) then \( x' \leftarrow \text{Shift-to-Zero}(x) \) and \( y' \leftarrow \text{Sign-to-Zero}(x) \)

14. else \( y' \leftarrow \text{Shift-to-Zero}(y) \) and \( x' \leftarrow \text{Sign-to-Zero}(y) \)

15. else if \((x \in \{-1_d, +1_d\} \text{ and } \text{weight}(y) = 1 \text{ and } \text{sgn}(x) \neq \text{sgn}(y)) \text{ or } \)

16. \((y \in \{-1_d, +1_d\} \text{ and } \text{weight}(x) = 1 \text{ and } \text{sgn}(y) \neq \text{sgn}(x))\) then

17. \( x' \leftarrow -0 \) and \( y' \leftarrow +0 \)

18. else

19. \( x' \leftarrow \text{Shift-to-Zero}(x) \) and \( y' \leftarrow \text{Shift-to-Zero}(y) \)
Protocols can become complex, even for $B \geq R$:

Fast and Exact Majority in Population Protocols

1. \( \text{weight}(x) = \begin{cases} |x| & \text{if } x \in \text{StrongStates or } x \in \text{WeakStates;} \\
1 & \text{if } x \in \text{IntermediateStates.} \end{cases} \)
2. \( \text{sgn}(x) = \begin{cases} 1 & \text{if } x \in \{+0, 1, -1, 3, 5, \ldots, m\}; \\
-1 & \text{otherwise.} \end{cases} \)
3. \( \text{value}(x) = \text{sgn}(x) \cdot \text{weight}(x) \)

/* Functions for rounding state interactions */
4. \( \phi(x) = -1 \) if \( x = -1; 1 \) if \( x = 1; x \), otherwise
5. \( R_1(k) = \phi(k \text{ if } k \text{ odd integer, } k - 1 \text{ if } k \text{ even}) \)
6. \( R_7(k) = \phi(k \text{ if } k \text{ odd integer, } k + 1 \text{ if } k \text{ even}) \)
7. \( \text{Shift-to-Zero}(x) = \begin{cases} -1_{j+1} & \text{if } x = -1_j \text{ for some index } j < d \\
1_{j+1} & \text{if } x = 1_j \text{ for some index } j < d \\
x & \text{otherwise.} \end{cases} \)
8. \( \text{Sign-to-Zero}(x) = \begin{cases} +0 & \text{if } \text{sgn}(x) > 0 \\
-0 & \text{otherwise.} \end{cases} \)

procedure \( \text{update}(x, y) \)
9. \( \text{if } (\text{weight}(x) > 0 \text{ and } \text{weight}(y) > 1) \text{ or } (\text{weight}(y) > 0 \text{ and } \text{weight}(x) > 1) \text{ then} \)
10. \( x' \leftarrow R_1\left(\frac{|\text{value}(x)| + |\text{value}(y)|}{2}\right) \text{ and } y' \leftarrow R_7\left(\frac{|\text{value}(x)| + |\text{value}(y)|}{2}\right) \)
11. \( \text{else if } \text{weight}(x) \cdot \text{weight}(y) = 0 \text{ and } \text{value}(x) + \text{value}(y) > 0 \text{ then} \)
12. \( \text{if } \text{weight}(x) \neq 0 \text{ then } x' \leftarrow \text{Shift-to-Zero}(x) \text{ and } y' \leftarrow \text{Sign-to-Zero}(x) \)
13. \( \text{else } y' \leftarrow \text{Shift-to-Zero}(y) \text{ and } x' \leftarrow \text{Sign-to-Zero}(y) \)
14. \( \text{else if } (x \in \{-1_d, +1_d\} \text{ and } \text{weight}(y) = 1 \text{ and } \text{sgn}(x) \neq \text{sgn}(y)) \text{ or } \)
15. \( (y \in \{-1_d, +1_d\} \text{ and } \text{weight}(x) = 1 \text{ and } \text{sgn}(y) \neq \text{sgn}(x)) \text{ then } \)
16. \( x' \leftarrow -0 \text{ and } y' \leftarrow +0 \)
17. \( \text{else } \)
18. \( x' \leftarrow \text{Shift-to-Zero}(x) \text{ and } y' \leftarrow \text{Shift-to-Zero}(y) \)
Our approach:

- Most protocols are naturally analyzed in “stages”: “milestones” until the protocol reaches consensus 0 or 1, depending on the input.
Our approach:

- Most protocols are naturally analyzed in “stages”: “milestones” until the protocol reaches consensus 0 or 1, depending on the input.
- Sound and complete methodology reflecting this idea: Stage Graphs.
Our approach:

• Most protocols are naturally analyzed in “stages”: “milestones” until the protocol reaches consensus 0 or 1, depending on the input
• Sound and complete methodology reflecting this idea:

Stage Graphs

• Stage graphs for $b = 0$ and $b = 1$, describing “milestones” from the initial configurations for which the output should be $b$
Our approach:

- Most protocols are naturally analyzed in “stages”: “milestones” until the protocol reaches consensus 0 or 1, depending on the input
- Sound and complete methodology reflecting this idea:

  **Stage Graphs**

  - Stage graphs for $b = 0$ and $b = 1$, describing “milestones” from the initial configurations for which the output should be $b$
  - SMT-based semi-algorithm for the automatic construction of stage graphs
Preliminaries: **Transition-based consensus**

- Split set $T$ of transitions into $T_0, T_1, T_\bot$.
- Run reaches **stable consensus** $b$ if from some moment on it only executes transitions of $T_b$.
- Equivalent to state-based consensus
Preliminaries: Transition-based consensus

- Split set $T$ of transitions into $T_0, T_1, T_\perp$.
- Run reaches stable consensus $b$ if from some moment on it only executes transitions of $T_b$.
- Equivalent to state-based consensus

Milestones: "killing" transitions

- A transition is dead at a configuration $C$ if no configuration reachable from $C$ enables it.
- Intuition: protocols make progress towards consensus by "killing" transitions, until all "survivors" in $T_0$ or all in $T_1$. 

Preliminaries: **Progress certificates**

Let \( C, C' \) be sets of configurations

- \( C \leadsto C' \): runs starting at \( C \) visit \( C' \) with probability 1
- **Certificate for** \( C \leadsto C' \): mapping \( f: C \to \mathbb{N} \) such that for every \( C \in C \setminus C' \) there exists \( C \overset{*}{\rightarrow} C' \) such that \( f(C) > f(C') \).
Let $C, C'$ be sets of configurations

- $C \rightsquigarrow C'$: runs starting at $C$ visit $C'$ with probability 1
- **Certificate for $C \rightsquigarrow C'$**: mapping $f: C \to \mathbb{N}$ such that for every $C \in C \setminus C'$ there exists $C \xrightarrow{*} C'$ such that $f(C) > f(C')$.
- **$k$-Bounded certificate for $C \rightsquigarrow C'$**: replace $C \xrightarrow{*} C'$ by $C \xrightarrow{k} C'$. 

**Easy but important**

If $C$ is inductive (closed under reachability): $C \rightsquigarrow C'$ iff there is a certificate for it.
Preliminaries: Progress certificates

Let $C, C'$ be sets of configurations

- $C \rightsquigarrow C'$: runs starting at $C$ visit $C'$ with probability 1
- Certificate for $C \rightsquigarrow C'$: mapping $f: C \rightarrow \mathbb{N}$ such that for every $C \in C \setminus C'$ there exists $C \xrightarrow{*} C'$ such that $f(C) > f(C')$.
- $k$-Bounded certificate for $C \rightsquigarrow C'$: replace $C \xrightarrow{*} C'$ by $C \xrightarrow{k} C'$.

Easy but important

If $C$ is inductive (closed under reachability):

$C \rightsquigarrow C'$ iff there is a certificate for it
If $\mathcal{C}$ is inductive (closed under reachability):

$\mathcal{C} \leadsto \mathcal{C}'$ iff there is a certificate for it
A stage graph for a given protocol, a given predicate, and a given \( b \in \{0, 1\} \) is a finite DAG satisfying:
A **stage graph** for a given protocol, a given predicate, and a given $b \in \{0, 1\}$ is a finite DAG satisfying:

1. The nodes of the DAG, called **stages**, are inductive sets of configurations (closed under reachability)
A **stage graph** for a given protocol, a given predicate, and a given $b \in \{0, 1\}$ is a finite DAG satisfying:

1. The nodes of the DAG, called **stages**, are inductive sets of configurations (closed under reachability)
2. Every $b$-initial configuration of the protocol belongs to some initial stage
A stage graph for a given protocol, a given predicate, and a given \( b \in \{0, 1\} \) is a finite DAG satisfying:

1. The nodes of the DAG, called stages, are inductive sets of configurations (closed under reachability)
2. Every \( b \)-initial configuration of the protocol belongs to some initial stage
3. For every non-terminal stage \( C \) with children \( C_1, \ldots, C_n \) there is a certificate for \( C \leadsto C_1 \cup \cdots \cup C_n \)
A stage graph for a given protocol, a given predicate, and a given $b \in \{0, 1\}$ is a finite DAG satisfying:

1. The nodes of the DAG, called stages, are inductive sets of configurations (closed under reachability)
2. Every $b$-initial configuration of the protocol belongs to some initial stage
3. For every non-terminal stage $C$ with children $C_1, \ldots, C_n$ there is a certificate for $C \rightsquigarrow C_1 \cup \cdots \cup C_n$
4. For every terminal stage $C$: every $C \in C$ enables only transitions of $T_b$
An example

Majority protocol \((R > B)\)

\[\begin{align*}
t_1: & \quad B, R \leftrightarrow b, b \\
t_2: & \quad B, r \leftrightarrow B, b \\
t_3: & \quad R, b \leftrightarrow R, r \\
t_4: & \quad b, r \leftrightarrow b, b
\end{align*}\]
An example

**Majority protocol** \((R > B)\)

\[
\begin{align*}
  t_1 & : \text{B, R} \leftrightarrow \text{b, b} & t_3 & : \text{R, b} \leftrightarrow \text{R, r} \\
  t_2 & : \text{B, r} \leftrightarrow \text{B, b} & t_4 & : \text{b, r} \leftrightarrow \text{b, b}
\end{align*}
\]

**Stage graph for** \(b = 0\)

\[
\begin{align*}
  R \leq B, b = 0 & \lor R + r > 0 \\
  \text{Cert.: } |R| + |B| \\
  R = 0, B > 0 & \quad B + R = 0, r > 0 \\
  \text{Cert.: } |b| & \quad \text{Cert.: } |b| \\
  B + b = 0
\end{align*}
\]
An example

Majority protocol \((R > B)\)

- \(t_1: \ B, R \leftrightarrow b, b\)
- \(t_2: \ B, r \leftrightarrow B, b\)
- \(t_3: \ R, b \leftrightarrow R, r\)
- \(t_4: \ b, r \leftrightarrow b, b\)

Stage graph for \(b = 0\)

- \(R \leq B, b = 0 \lor R + r > 0\)
  - Cert.: \(|R| + |B|\)
- \(R = 0, B > 0\)
  - Cert.: \(|b|\)
- \(B + R = 0, r > 0\)
  - Cert.: \(|b|\)
- \(B + b = 0\)

Stage graph for \(b = 1\)

- \(R > B\)
  - Cert.: \(|R| + |B|\)
- \(R > 0, B = 0\)
  - Cert.: \(|b|\)
- \(R + r = 0\)
**Soundness**

If a protocol has stage graphs for a predicate \( \varphi \) and both 0 and 1, then the protocol computes \( \varphi \).

**Proof.**

Easy.

Show that executions “go down” the stage graph w.p.1 till they get “trapped” in a bottom stage.
A **Presburger stage graph** is a stage graph whose nodes are Presburger sets and whose certificates are $k$-bounded Presburger certificates $(f(C) = a$ iff $\psi(C, a) \equiv \text{true}$) for some $k$.

**Completeness**

If a protocol computes $\varphi$, then it has Presburger stage graphs for $\varphi$ and both 0 and 1.

**Proof.**

Very hard.

Initial stage: Inductive Presburger “envelope” of the $b$-initial configurations.

Final stage: set of all $b$-stable consensuses.
### Decidability

It is decidable if a given DAG of Presburger sets and certificates is a Presburger stage graph.

### Proof.

Follows from properties of Presburger sets:

- Inductivity of Presburger sets is decidable (because Presburger arithmetic is)
- Whether a Presburger function is a $k$-bounded Presburger certificate is decidable
Alternative algorithm for decidability of correctness:

- Two semi-decision algorithms
- For non-correctness: enumerate all initial configurations and check convergence to the right value
- For correctness: enumerate all DAGs of Presburger sets and functions, and check if they are Presburger stage graphs for 0 or for 1
Computing inductive envelopes: The liquid approximation
Computing inductive envelopes: The liquid approximation
Computing inductive envelopes: **The liquid approximation**
Fluid agents in action

\[(A, B_1) \mapsto (D, B_2)\]
\[(A, C_1) \mapsto (D, C_2)\]
\[(B_1, B_2) \mapsto (D, D)\]
\[(C_1, C_2) \mapsto (D, D)\]

The liquid reachability set of a Presburger set is an effectively computable Presburger set.
Fluid agents in action

\[(A, B_1) \mapsto (D, B_2)\]
\[(A, C_1) \mapsto (D, C_2)\]
\[(B_1, B_2) \mapsto (D, D)\]
\[(C_1, C_2) \mapsto (D, D)\]

**Theorem**

The liquid reachability set of a Presburger set is an effectively computable Presburger set.
Computing the children of a non-terminal stage $S$

**Goal:** $S \leadsto S_1 \cup \cdots \cup S_n$ with more dead transitions

**Input:** $S$

$U := \text{AsDead}(S)$

**if** $U \neq \emptyset$ **then**

**output** $\text{IndEnvelope}(U, S)$

**else**

**output** $\text{Split}(S)$
Computing the children of a non-terminal stage $S$

**Goal:** $S \rightsquigarrow S_1 \cup \cdots \cup S_n$ with more dead transitions

**Input:** $S$

$U := \text{AsDead}(S)$

**if** $U \neq \emptyset$ **then**

**output** $\text{IndEnvelope}(U, S)$

**else**

**output** $\text{Split}(S)$

Returns a set of transitions that will die almost surely from **any** configuration of $S$.
Computing the children of a non-terminal stage $S$

Goal: $S \rightsquigarrow S_1 \cup \cdots \cup S_n$ with more dead transitions

**Input:** $S$

$U := \text{AsDead}(S)$

**if** $U \neq \emptyset$ **then**

**output** $\text{IndEnvelope}(U, S)$

**else**

**output** $\text{Split}(S)$

Overapproximates the configurations of $S$ at which $U$ is dead
Computing the children of a non-terminal stage $S$

Goal: $S \rightsquigarrow S_1 \cup \cdots \cup S_n$ with more dead transitions

Input: $S$

$U := \text{AsDead}(S)$

if $U \neq \emptyset$ then

output $\text{IndEnvelope}(U, S)$

else

output $\text{Split}(S)$

Attempts to split $S$ into stages with more dead transitions
Computing the children of a non-terminal stage $S$

Goal: $S \rightsquigarrow S_1 \cup \cdots \cup S_n$ with more dead transitions

Input: $S$

$U := \text{AsDead}(S)$

if $U \neq \emptyset$ then

\textbf{output} $\text{IndEnvelope}(U, S)$

else

\textbf{output} $\text{Split}(S)$
Implementing AsDead($\mathcal{S}$): **Kirchoff’s equations**

Transition $t \implies$ offset $\Delta(t)$

Examples:

- $t$: $q_1, q_2 \mapsto q_2, q_3 \implies \Delta(t) = (-1, 0, 1)$
- $t$: $q_1, q_2 \mapsto q_3, q_3 \implies \Delta(t) = (-1, -1, 2)$

We have: if $C \xrightarrow{t} C'$ then $C' = C + \Delta(t)$
Implementing AsDead($\mathcal{S}$): Kirchoff’s equations

Transition $t \implies$ offset $\Delta(t)$

Examples:
- $t: q_1, q_2 \mapsto q_2, q_3 \implies \Delta(t) = (-1, 0, 1)$
- $t: q_1, q_2 \mapsto q_3, q_3 \implies \Delta(t) = (-1, -1, 2)$

We have: if $C \xrightarrow{t} C'$ then $C' = C + \Delta(t)$

Transition sequence $w = t_1 \ldots t_n$

$$\implies \text{offset } \Delta(w) = \sum_{i=1}^{n} \Delta(t_i) = \sum_{t \in T} \#(t, w) \cdot \Delta(t)$$

Example: if $C \xrightarrow{t_1 t_2 t_1} C'$ then $C' = C + 2 \cdot \Delta(t_1) + \Delta(t_2)$

We have: if $C \xrightarrow{w} C'$ then $C' = C + \Delta(w)$
Implementing AsDead($\mathcal{S}$): Kirchoff’s equations

If transition $u$ can occur infinitely often

$\implies$ there is $C \xrightarrow{w} C$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\Delta(w) = 0$ s.t. $w$ contains $u$
Implementing AsDead($S$): Kirchoff’s equations

If transition $u$ can occur infinitely often

$\implies$ there is $C \xrightarrow{w} C$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\Delta(w) = 0$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\sum_{t \in T} #(t, w) \cdot \Delta(t) = 0$ s.t. $#(u, w) \geq 1$
Implementing $\text{AsDead}(S)$: Kirchoff’s equations

If transition $u$ can occur infinitely often

$\implies$ there is $C \xrightarrow{w} C$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\Delta(w) = 0$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\sum_{t \in T} \#(t, w) \cdot \Delta(t) = 0$ s.t. $\#(u, w) \geq 1$
Implementing AsDead(\(S\)): Kirchoff’s equations

If transition \(u\) can occur infinitely often

\[\implies\text{there is } C \xrightarrow{w} C\text{ s.t. } w\text{ contains } u\]

\[\implies\text{there is } w\text{ with } \Delta(w) = 0\text{ s.t. } w\text{ contains } u\]

\[\implies\text{there is } w\text{ with } \sum_{t \in T} \#(t, w) \cdot \Delta(t) = 0\text{ s.t. } \#(u, w) \geq 1\]
Implementing AsDead(\(S\)):

Kirchoff’s equations

If transition \(u\) can occur infinitely often

\[\implies\] there is \(C \xrightarrow{w} C\) s.t. \(w\) contains \(u\)

\[\implies\] there is \(w\) with \(\Delta(w) = 0\) s.t. \(w\) contains \(u\)

\[\implies\] there is \(w\) with \(\sum_{t \in T} \#(t, w) \cdot \Delta(t) = 0\) s.t. \(\#(u, w) \geq 1\)

\[\implies\] there are coefficients \(n_t\) for every \(t \in T\) with

\[\sum_{t \in T} n_t \cdot \Delta(t) = 0 \quad \text{and} \quad n_u \geq 1\]
Implementing AsDead($\mathcal{S}$): Kirchoff’s equations

If transition $u$ can occur infinitely often

$\implies$ there is $C \xrightarrow{w} C$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\Delta(w) = 0$ s.t. $w$ contains $u$

$\implies$ there is $w$ with $\sum_{t \in T} #(t, w) \cdot \Delta(t) = 0$ s.t. $#(u, w) \geq 1$

$\implies$ there are coefficients $n_t$ for every $t \in T$ with

$$\sum_{t \in T} n_t \cdot \Delta(t) = 0 \quad \text{and} \quad n_u \geq 1$$

Kirchoff’s equations
(unknowns: $\{n_t \mid t \in T\}$)
Implementing AsDead(S): Kirchoff’s equations

If transition \( u \) can occur infinitely often

\[ \implies \text{there is } C \xrightarrow{w} C \text{ s.t. } w \text{ contains } u \]

\[ \implies \text{there is } w \text{ with } \Delta(w) = 0 \text{ s.t. } w \text{ contains } u \]

\[ \implies \text{there is } w \text{ with } \sum_{t \in T} \#(t, w) \cdot \Delta(t) = 0 \text{ s.t. } \#(u, w) \geq 1 \]

\[ \implies \text{there are coefficients } n_t \text{ for every } t \in T \text{ with} \]

\[ \sum_{t \in T} n_t \cdot \Delta(t) = 0 \text{ and } n_u \geq 1 \]

Kirchoff’s equations unsatisfiable

\[ \implies t \text{ cannot occur infinitely often from any configuration} \]
Implementing AsDead(S): Layers

A layer of a protocol is a set \( L \) of transitions such that for every configuration \( C \) (reachable or not):

- all executions from \( C \) containing only transitions of \( L \) are finite
- if all transitions of \( L \) are disabled at \( C \), then they cannot be re-enabled by any sequence \( w \in (T \setminus L)^* \).

If \( L \) is a layer, then from any configuration all transitions of \( L \) eventually die
Implementing AsDead($S$): Layers

A layer of a protocol is a set $L$ of transitions such that for every configuration $C$ (reachable or not):

- all executions from $C$ containing only transitions of $L$ are finite
- if all transitions of $L$ are disabled at $C$, then they cannot be re-enabled by any sequence $w \in (T \setminus L)^*$.

If $L$ is a layer, then from any configuration all transitions of $L$ eventually die.

There exists a set of integer linear constraints whose solutions correspond to the possible layers of the protocol → finding a layer is in NP.
Implementing IndEnvelope($U, S$)

Recall: $IndEnvelope(U, S)$ overapproximates the configurations reachable from $S$ at which $U$ is dead.

Computable as intersection of:

- overapproximation of the configurations reachable from $S$ (overapproximation is necessary)
- overapproximation of the configurations at which $U$ is dead (overapproximation is optional)
Implementing IndEnvelope\((U, S)\)

Set of configurations reachable from \(S\)

Overapproximated using the liquid approximation

Set of configurations at which \(U\) is dead
Implementing IndEnvelope\((U, S)\)

Set of configurations reachable from \(S\)
Overapproximated using the liquid approximation

Set of configurations at which \(U\) is dead
\(\text{Dead}(U)\): configurations from which \(U\) cannot be enabled
\(\text{En}(U)\): configurations enabling some transition of \(U\).

\[
\text{Dead}(U) = \text{pre}^*(\text{En}(U))
\]
Implementing IndEnvelope($U, S$)

Set of configurations reachable from $S$

Overapproximated using the liquid approximation

Set of configurations at which $U$ is dead

$Dead(U)$: configurations from which $U$ cannot be enabled
$En(U)$: configurations enabling some transition of $U$.

$$Dead(U) = \text{pre}^*(En(U))$$

Observation: $\text{pre}^*(En(U))$ is upward-closed

$Dead(U)$ is downward-closed
Implementing IndEnvelope\((U, S)\)

Set of configurations reachable from \(S\)

Overapproximated using the liquid approximation

Set of configurations at which \(U\) is dead

\(\text{Dead}(U)\): configurations from which \(U\) cannot be enabled

\(\text{En}(U)\): configurations enabling some transition of \(U\).

\[ \text{Dead}(U) = \text{pre}^*(\text{En}(U)) \]

Observation: \(\text{pre}^*(\text{En}(U))\) is upward-closed

\(\text{Dead}(U)\) is downward-closed

\(\implies\) both are Presburger
Implementing IndEnvelope\((U, S)\)

Set of configurations reachable from \(S\)

Overapproximated using the liquid approximation

Set of configurations at which \(U\) is dead

**Proposition**

\(\text{Dead}(U)\) has finitely many maximal elements, and they can be computed using a symbolic backward reachability algorithm.
Implementing IndEnvelope$(U, S)$

Set of configurations reachable from $S$
Overapproximated using the liquid approximation

Set of configurations at which $U$ is dead

**Proposition**

$\text{Dead}(U)$ has finitely many maximal elements, and they can be computed using a symbolic backward reachability algorithm.

$\Rightarrow \text{Dead}(U)$ is effectively Presburger
Some experimental results

Intel Core i7-4810MQ CPU and 16 GB of RAM.

| Protocol                         | Predicate                      | $|Q|$ | $|T|$ | Time[s] |
|----------------------------------|--------------------------------|-----|-----|---------|
| Majority[1]                      | $x \geq y$                     | 4   | 4   | 0.1     |
| Approx. Majority[2]              | Bug                            | 3   | 4   | 0.1     |
| Broadcast[3]                     | $x_1 \lor \ldots \lor x_n$     | 2   | 1   | 0.1     |
| Threshold[4]                     | $\sum_i \alpha_i x_i < c$      | 76  | 2185| 445     |
| Remainder[5]                     | $\sum_i \alpha_i x_i \bmod 70 = 1$ | 72  | 2555| 3176    |
| Sick ninjas[6]                   | $x \geq 50$                    | 51  | 1275| 181     |
| Poly-log sick ninjas             | $x \geq 8 \cdot 10^4$          | 66  | 244 | 12      |
| Av. and Conquer[7]               | $x \geq y$                     | 32  | 528 | 1866    |

[1] Draief et al., 2012  
[3] Clément et al., 2011  
[4][5] Angluin et al., 2006  
[6] Chatzigiannakis et al., 2010  
Some experimental results

Intel Core i7-4810MQ CPU and 16 GB of RAM.

| System                        | |Q| | |T| | Time[s] |
|-------------------------------|-----|-----|--------|
| Israeli-Jalfon [20]           | 40  | 80  | 5.4    |
| Israeli-Jalfon [70]           | 140 | 280 | 2537   |
| Herman [11]                   | 22  | 22  | 1.2    |
| Herman [91]                   | 142 | 142 | 741    |
| Burns [6]                     | 27  | 275 | 1074   |
Some experimental results

Intel Core i7-4810MQ CPU and 16 GB of RAM.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Time bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority with tie break</td>
<td>$e^{O(n \log n)}$</td>
</tr>
<tr>
<td>Majority no tie break</td>
<td>$O(n^2 \log n)$</td>
</tr>
<tr>
<td>Threshold</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Remainder</td>
<td>$O(n^2 \log n)$</td>
</tr>
<tr>
<td>Sick ninjas</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Poly-log sick ninjas</td>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>Av. and Conquer</td>
<td>$O(n^2 \log n)$</td>
</tr>
</tbody>
</table>
Peregrine: Haskell + Microsoft Z3 + JavaScript

peregrine.model.in.tum.de

• Design of protocols
• Manual and automatic simulation
• Statistics of properties such as termination time
• Automatic verification of correctness
• More to come!
Population protocols can be formally analyzed automatically:

- Verification of correctness
- Analysis of expected termination time
- Tool support
THANK YOU!