

On the Self-Stabilization of Dijkstra's Asynchronous Token Circulation

Asma Khoualdia, Sami Cherif, Stéphane Devismes, Léo Robert

CP 2026



1 Preliminaries

- Asynchronous Token Circulation
- Propositional Satisfiability

2 Formal Modeling

- Variables and Notations
- Modeling the Execution
- Convergence Modeling
- Divergence Modeling
- Offset-Equivalence
- Beyond The Unfair Distributed Daemon

3 Experimental Evaluation

- Experimental Protocol
- Results

4 Conclusion

Distributed Algorithms

Definition

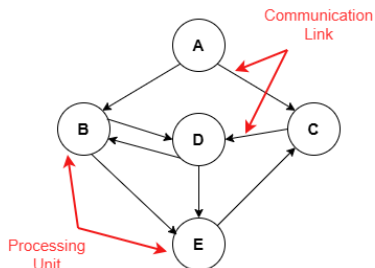
A set of rules executed locally by each process to solve a global task through only local interactions between processes.



*"In computer science, a distributed system [...] is a set of **autonomous** processing units **interconnected** with one another."*

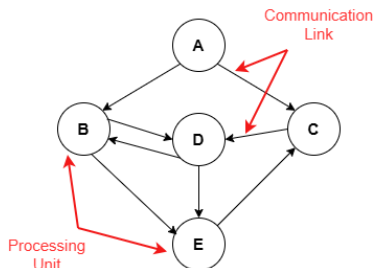
— Gérard Tel

Distributed Algorithms



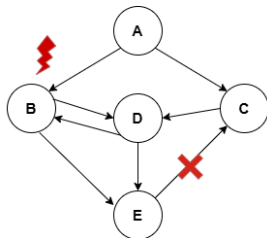
- **Autonomous:** Each machine has its own control → local memory and local program.

Distributed Algorithms



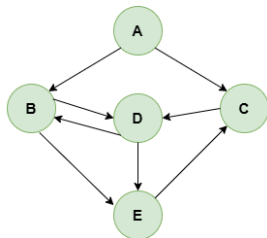
- **Autonomous:** Each machine has its own control → local memory and local program.
- **Interconnected:** Each machine can exchange information through communication links.

Why Self-stabilization?



- **Transient faults:** messages may be lost, memory corruption.
- **Arbitrary state:** the system can start in any state.

Why Self-stabilization?



- **Algorithm execution:** despite faults, the algorithm keeps running.
- **Correct behavior:** the system converges to a correct behavior.

Self-Stabilization

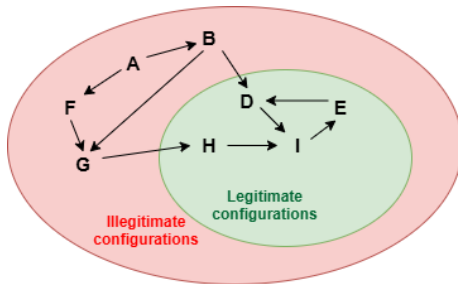
Definition

An algorithm is self-stabilizing if, from any initial state, it guarantees a return in finite time to a legitimate state without external intervention.

Self-Stabilization

Definition

An algorithm is self-stabilizing if, from any initial state, it guarantees a return in finite time to a legitimate state without external intervention.

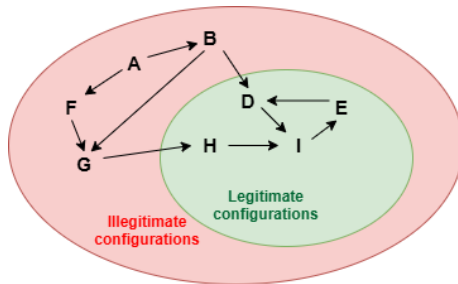


- Two properties: convergence,

Self-Stabilization

Definition

An algorithm is self-stabilizing if, from any initial state, it guarantees a return in finite time to a legitimate state without external intervention.



- Two properties: convergence, closure

Asynchronism and Daemons

- **Asynchronous system:** a system in which not all machines or processes operate at the same speed.
- **Unfair distributed daemon:** at each step, an arbitrary non-empty subset of enabled processes is activated.



Analyzing Self-Stabilization of Synchronous Unison via Propositional Satisfiability

Asma Khoualdia, Sami Cherif, Stéphane Devismes, Léo Robert.

In CP 2025, 10-15 August 2025, Glasgow, Scotland

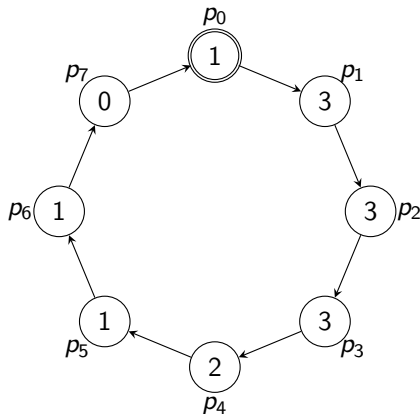
Dijkstra's Token Circulation Algorithms

"I thought that in Dijkstra 1974, I had published three solutions, but later I learned that I had also published three problems."

Edsger W. Dijkstra, A Belated Proof of Self-Stabilization, 1986.



Dijkstra's K -state Algorithm

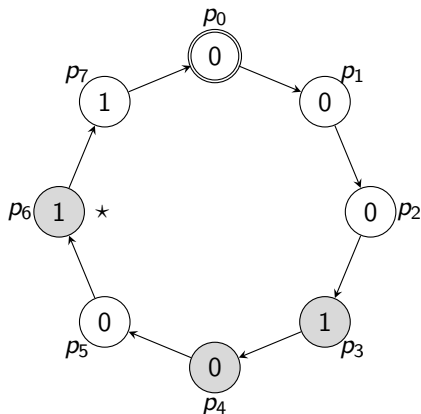


- Rooted directed ring
- Each process has a state value $v \in \{0, \dots, K - 1\}$
- $\text{pred}(p) = p - 1 \mod n$
- Asynchronous setting

Restricted Daemons

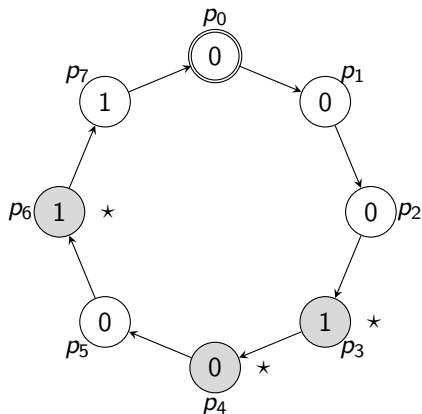
Sequential Daemon

- Activates **exactly one** enabled process per step.



Synchronous Daemon

- Activates **every** enabled process at each step.



- **Asynchronous self-stabilization analysis**
 - Formal modeling of Dijkstra's K -state algorithm executions and behavior.
- **Tight characterization of the stabilization bound**
 - Study of the **tightness** of the bound $K \geq n$.
- **Analysis of unresolved daemon scenarios**
 - **non-synchronous** executions for $K = n - 1$,
 - **non-central** executions for $K < n - 1$.
- **Model optimization**
 - Introduction of offset-based symmetry breaking to reduce redundant initial configurations.

Predicate $\rightarrow$ Action Model in Dijkstra's Algorithm

Execution Rule:

IF **predicate** $\rightarrow$ THEN execute **action**

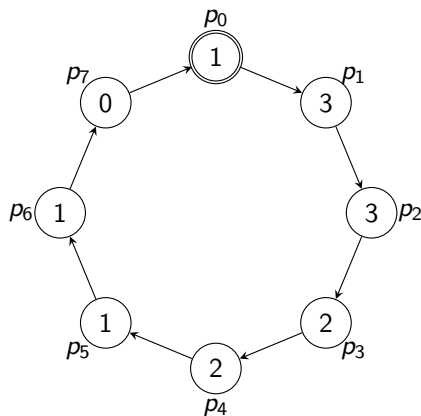
Token:

- $Token(p_0) \stackrel{def}{=} (p_0.v = pred(p_0).v)$
- $Token(p) \stackrel{def}{=} (p.v \neq pred(p).v), p \neq p_0$

Execution:

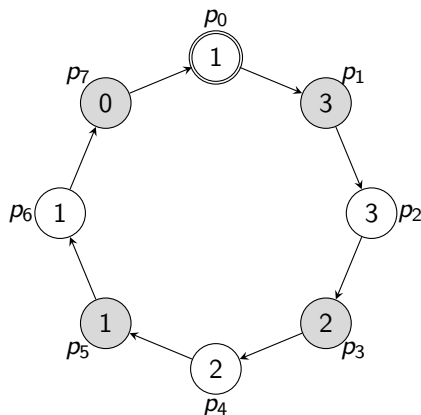
- $Token(p_0) \rightarrow p_0.v \leftarrow (p_0.v + 1) \bmod K$
- $Token(p) \rightarrow p.v \leftarrow pred(p).v, p \neq p_0$

Example: Convergence



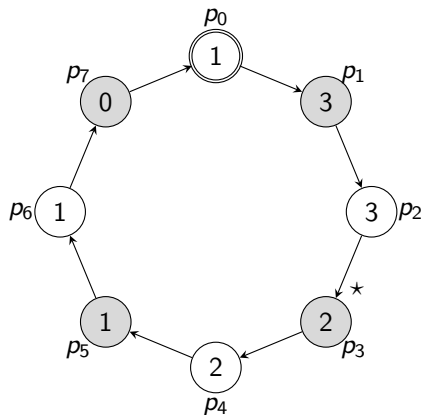
- p_0 is the root process
- $p \neq p_0$ are the non-root processes
- Initial configuration of size 8 and $K = 4$

Example



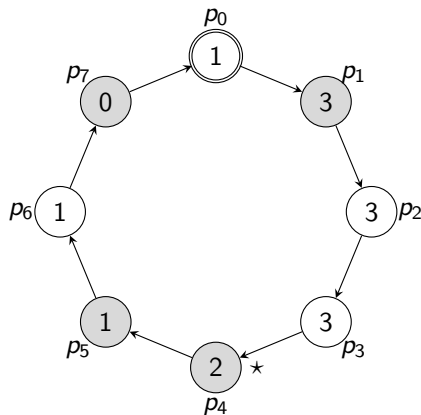
- Initial configuration γ_0
- Processes p_1, p_3, p_5 and p_7 hold the token
- These processes are then **enabled**
- Processes p_0, p_2, p_4 and p_6 do not hold the token
- These processes are then **disabled**

Example



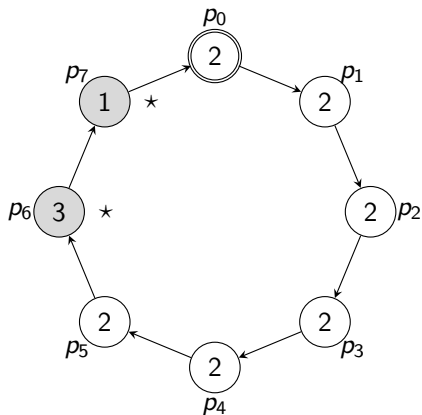
- Process p_3 is chosen by the daemon to perform the **action**
- Process p_3 was then **activated** by the daemon

Example: Convergence



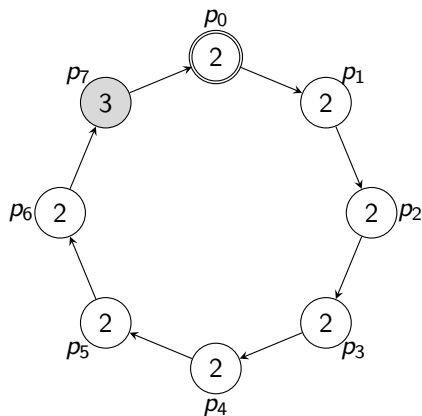
- Configuration γ_1
- Process p_3 updated its value
- As p_3 is a non-root process its value is update by taking its predecessor's value

Example



- Configuration γ_9
- Enabled processes p_6, p_7 are **activated** by the daemon

Example

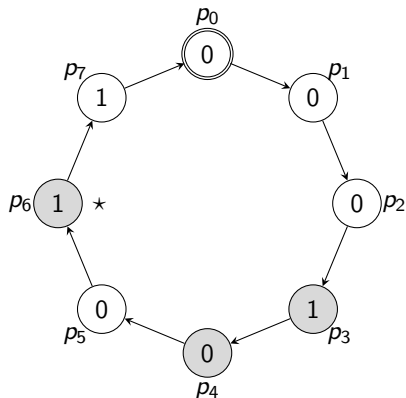


- Configuration γ_{10}
- Only process p_7 holds the token
- Only one token is circulating

The algorithm **converges** when, from any initial configuration $\gamma_0 \in \Gamma_{n,K}$:

- **Every possible** asynchronous execution reaches a **legitimate** configuration
- Legitimate configuration $\equiv$ where there is a **unique token**
- Within a **finite** number of iterations

Example: Divergence



Initial divergent configuration γ_0 in an 8-node ring ($K = 2$)

- Configuration γ_0 and γ_{12} are identical
- In the initial configuration γ_0 three tokens are circulating
→ Illegitimate configuration
- The system will never reach a legitimate configuration

The algorithm **diverges** if there exists an initial configuration $\gamma_0 \in \Gamma_{n,K}$ such that:

- Two distinct processes simultaneously hold the token,
- There exists an infinite sequence of configurations containing at least two tokens.

Propositional Satisfiability

- **Literal:** A Boolean variable $l = x$ or its negation $l = \neg x$
- **Clauses:** A disjunction of literals, $c = l_1 \vee \dots \vee l_n$
- **CNF:** A conjunction of clauses, $\phi = c_1 \wedge \dots \wedge c_m$
- **Satisfiability (SAT):** Find an assignment of Boolean variables that satisfies a given CNF formula

Objective

Model the behavior of the algorithm using propositional satisfiability:

- The self-stabilisation or its absence are encoded as SAT formulas
- We adopt **Bounded Model Checking (BMC)**: the system is analyzed up to a finite execution bound

A directed rooted ring $G = (V, E)$ with $n \geq 3$ processes.

- **Processes:** indexed as $V = \{0, \dots, n-1\}$, where p_0 is the root.
- **Predecessor:** each process $p \in V$ has a unique predecessor $\text{pred}(p) = (p-1) \bmod n$.
- **Values:** each process holds a value in $M = \{0, \dots, K-1\}$.
- **Steps:** we consider the first t_f configurations, indexed by $T = \{0, \dots, t_f-1\}$.

- **State Variables:** $st_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Authorization Variables:** $aut_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Token Variables:** $tok_{p,t}$ where $p \in V, t \in T$.
- **Activation Variables:** $act_{p,t}$ where $p \in V, t \in T$.
- **Cycle Variables:** cyc_t where $t \in T$.
- **State Similarity Variables:** $sim_{p,t,v}$ where $p \in V, t \in T, v \in M$.

State Uniqueness

$$\sum_{v \in M} st_{p,t,v} = 1, \quad \forall p \in V, t \in T \quad (1)$$

- (1) ensures that a process has **exactly one state value** at a given time.

Binomial CNF encoding:

$$\bigwedge_{\substack{v, v' \in M \\ v < v'}} (\overline{st_{p,t,v}} \vee \overline{st_{p,t,v'}}) \quad \forall p \in V, t \in T$$
$$\bigvee_{v \in M} st_{p,t,v} \quad \forall p \in V, t \in T$$

- The binomial encoding yields $O(n \cdot t_f \cdot K^2)$ clauses.

- **State Variables:** $st_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Authorization Variables:** $aut_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Token Variables:** $tok_{p,t}$ where $p \in V, t \in T$.
- **Activation Variables:** $act_{p,t}$ where $p \in V, t \in T$.
- **Cycle Variables:** cyc_t where $t \in T$.
- **State Similarity Variables:** $sim_{p,t,v}$ where $p \in V, t \in T, v \in M$.

$$aut_{p,t,v} \leftrightarrow (st_{p,t,v} \wedge st_{pred(p),t,v}) \quad \forall p \in V, t \in T, v \in M \quad (2)$$

$$tok_{0,t} \rightarrow \bigvee_{v \in M} aut_{0,t,v}, \quad \forall t \in T \quad (3a)$$

$$tok_{p,t} \rightarrow \bigvee_{v \in M} aut_{p,t,v}, \quad \forall p \in V \setminus \{0\}, t \in T, v \in M \quad (3b)$$

- (2) encodes authorisation variable semantics ensuring that each process holds the same value as its predecessor at configuration t .
- (3a) and (3b) encode token possession semantics for root and other processes at configuration t .
- These constraints generate a number of clauses in $O(n * t_f * K)$

- **State Variables:** $st_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Authorization Variables:** $aut_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Token Variables:** $tok_{p,t}$ where $p \in V, t \in T$.
- **Activation Variables:** $act_{p,t}$ where $p \in V, t \in T$.
- **Cycle Variables:** cyc_t where $t \in T$.
- **State Similarity Variables:** $sim_{p,t,v}$ where $p \in V, t \in T, v \in M$.

Process Activation

$$act_{p,t} \rightarrow tok_{p,t} \quad \forall p \in V, t \in T \setminus \{t_f - 1\} \quad (4a)$$

$$\bigvee_{p \in V} act_{p,t}, \quad \forall t \in T \setminus \{t_f - 1\} \quad (4b)$$

- (4a) ensures that an activated process must be enabled.
- (4b) ensures that the set of processes activated at each stage must be non-empty
- These constraints generate a number of **clauses** in $O(n * t_f)$

A process can be in one of the following three situations:

- (i) **Disabled**: the process retains its current state.
- (ii) **Enabled** but **not activated**: the state remains unchanged.
- (iii) **Enabled** and **activated**: the value is updated according to the algorithm rules.

State Update: Disabled Processes

$$\overline{aut_{0,t,v}} \wedge st_{0,t,v} \rightarrow st_{0,t+1,v}, \quad \forall t \in T \setminus \{t_f - 1\}, v \in M \quad (5a)$$

$$aut_{p,t,v} \rightarrow st_{p,t+1,v}, \quad \forall p \in V, t \in T \setminus \{t_f - 1\}, v \in M \quad (5b)$$

- (5a) and (5b) ensure that a disabled process maintains its state in the next configuration.
- These constraints generate a number of **clauses** in $O(n * t_f * K)$

State Update: Enabled and Inactivated Processes

$$st_{p,t,v} \wedge \overline{act_{p,t}} \rightarrow st_{p,t+1,v}, \quad \forall p \in V, t \in T \setminus \{t_f - 1\}, v \in M \quad (6)$$

- (6) ensures that each process which is not selected by the daemon maintains its state.
- These constraints generate a number of **clauses** in $O(n * t_f * K)$

State Update: Enabled and Activated Processes

$$aut_{0,t,v} \wedge act_{0,t} \rightarrow st_{0,t+1,(v+1) \bmod K}, \quad \forall t \in T, v \in M \quad (7a)$$

$$\overline{aut_{p,t,v}} \wedge st_{pred(p),t,v} \wedge act_{p,t} \rightarrow st_{p,t+1,v}, \forall p \in V, t \in T, v \in M \quad (7b)$$

- **Root process:** if activated, it updates its value by incrementing it modulo K .
- **Non-root processes:** if a process has a predecessor with a different value and is activated, then it adopts that value at the next step.
- Bounded by $O(n * t_f * K)$ Clauses

$$\sum_{p \in V} tok_{p, t_f - 1} \geq 2 \quad (8)$$

- The CNF formula ϕ (from constraints 1-8) is:
 - **unsatisfiable** $\Rightarrow$ the algorithm converges.
 - **satisfiable** $\Rightarrow$ there exists an initial configuration λ_0 that does not converge.
- (8) ensures that the final configuration violates convergence by enforcing the existence of at least two processes that simultaneously hold the token.

- **State Variables:** $st_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Authorization Variables:** $aut_{p,t,v}$ where $p \in V, t \in T, v \in M$.
- **Token Variables:** $tok_{p,t}$ where $p \in V, t \in T$.
- **Activation Variables:** $act_{p,t}$ where $p \in V, t \in T$.
- **Cycle Variables:** cyc_t where $t \in T$.
- **State Similarity Variables:** $sim_{p,t,v}$ where $p \in V, t \in T, v \in M$.

$$\bigvee_{t \in T \setminus \{0\}} \text{cyc}_t \quad (9a)$$

$$\sum_{p \in V} \text{tok}_{p,0} \geq 2 \quad (9b)$$

- (9a) : existence of a **cycle** $(0, t)$ where $t \in T$;
- (9b) : imposes that the initial configuration is illegitimate

$$cyc_t \rightarrow \bigvee_{v \in M} sim_{p,t,v}, \quad \forall t \in T \setminus \{0\}, p \in V \quad (10a)$$

$$sim_{p,t,v} \rightarrow st_{p,0,v} \wedge st_{p,t,v}, \quad \forall t \in T \setminus \{0\}, p \in V, v \in M \quad (10b)$$

- (9a) and (9b) encode the semantics of cycle and state similarity variables.
- Clauses in $O(n \times t_f \times K)$.
- The CNF formula induced by these constraints is **satisfiable iff the algorithm diverges within t_f configurations**.

Reducing Redundant Configurations: Offset-Equivalence

$$st_{0,0,0} \tag{11}$$

- Similar behavior for any pair of initial configurations in the same equivalence class under offset-equivalence.
- Two configurations λ and λ' are equivalent if:
 $\exists c \in 0, \dots, K-1, \quad \lambda' = \lambda^{+c}$
- Analyze only one representative per equivalence class instead of all redundant configurations.

Beyond The Unfair Distributed Daemon

- **Synchronous:** activates all enabled processes.

$$tok_{p,t} \rightarrow act_{p,t} \quad (12)$$

- **Non-synchronous:** doesn't activate at least one enabled process.

$$\bigvee_{p \in V} inact_p \quad (13a)$$

$$(\overline{inact_p} \vee tok_{p,0}) \bigwedge (\overline{inact_p} \vee \overline{act_{p,0}}), \quad \forall p \in V \quad (13b)$$

- **Non-sequential:** activates at least two enabled processes.

$$\bigvee_{p,p' \in V \text{ s.t. } p < p'} twoact_{p,p'} \quad (14a)$$

$$(\overline{twoact_{p,p'}} \vee act_{p,0}) \wedge (\overline{twoact_{p,p'}} \vee act_{p',0}), \quad \forall p, p' \in V \text{ with } p < p'. \quad (14b)$$

Experimental Protocol

- Models encoded in Python with PySAT¹.
- Solver used: **Cadical**.
- Time limit: **2h** per instance.
- **448** instances generated:
 - Sizes $n = 3..10$
 - Periods $K = 2..15$
- Models: Initial model (INI) and Offset-Elimination (OE) model

- Number of steps: $t_f = \frac{3n(n-1)}{2} 2$

¹<https://pysathq.github.io/>

²The threshold corresponds to the worst-case stabilization time bound of the K -state algorithm.

The K -state Algorithm Analysis

15	✓	✓	✓	✓	✓	⊖	-	-
14	✓	✓	✓	✓	✓	⊖	-	-
13	✓	✓	✓	✓	✓	⊖	-	-
12	✓	✓	✓	✓	✓	⊖	-	-
11	✓	✓	✓	✓	✓	⊖	-	-
10	✓	✓	✓	✓	✓	⊖	-	-
9	✓	✓	✓	✓	✓	⊖	-	⊖
8	✓	✓	✓	✓	✓	⊖	⊖	⊖
7	✓	✓	✓	✓	✓	⊖	⊖	⊖
6	✓	✓	✓	✓	⊖	⊖	⊖	⊖
5	✓	✓	✓	⊖	⊖	⊖	⊖	⊖
4	✓	✓	⊖	⊖	⊖	⊖	⊖	⊖
3	✓	⊖	⊖	⊖	⊖	⊖	⊖	⊖
2	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖
$K \backslash n$	3	4	5	6	7	8	9	10

(✓): algorithm converges (⊖): algorithm diverges (⊖): algorithm proven non-divergent within the set number of configurations (-): unresolved instance

Analysis

15	✓	✓	✓	✓	✓	⊖	-	-
14	✓	✓	✓	✓	✓	⊖	-	-
13	✓	✓	✓	✓	✓	⊖	-	-
12	✓	✓	✓	✓	✓	⊖	-	-
11	✓	✓	✓	✓	✓	⊖	-	-
10	✓	✓	✓	✓	✓	⊖	-	-
9	✓	✓	✓	✓	✓	⊖	-	⊖
8	✓	✓	✓	✓	✓	⊖	⊖	⊖
7	✓	✓	✓	✓	✓	⊖	⊖	⊖
6	✓	✓	✓	✓	✓	⊖	⊖	⊖
5	✓	✓	✓	⊖	⊖	⊖	⊖	⊖
4	✓	✓	⊖	⊖	⊖	⊖	⊖	⊖
3	✓	⊖	⊖	⊖	⊖	⊖	⊖	⊖
2	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖
$K \backslash n$	3	4	5	6	7	8	9	10

- All behaviors for $n \leq 7$ were verified
- Showing convergence consistent with theoretically proven results

15	✓	✓	✓	✓	✓	⊖	-	-
14	✓	✓	✓	✓	✓	⊖	-	-
13	✓	✓	✓	✓	✓	⊖	-	-
12	✓	✓	✓	✓	✓	⊖	-	-
11	✓	✓	✓	✓	✓	⊖	-	-
10	✓	✓	✓	✓	✓	⊖	-	-
9	✓	✓	✓	✓	✓	⊖	-	⊖
8	✓	✓	✓	✓	✓	⊖	⊖	⊖
7	✓	✓	✓	✓	✓	⊖	⊖	⊖
6	✓	✓	✓	✓	✓	⊖	⊖	⊖
5	✓	✓	✓	⊖	⊖	⊖	⊖	⊖
4	✓	✓	⊖	⊖	⊖	⊖	⊖	⊖
3	✓	⊖	⊖	⊖	⊖	⊖	⊖	⊖
2	⊖	⊖	⊖	⊖	⊖	⊖	⊖	⊖
$K \backslash n$	3	4	5	6	7	8	9	10

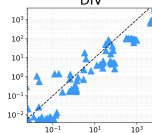
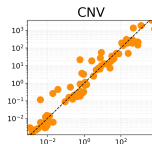
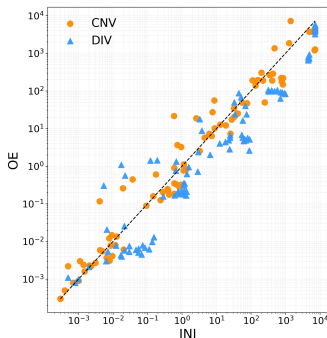
- All behaviors for $n \leq 7$ were verified
- Showing convergence consistent with theoretically proven results
- The model exhibits non divergence for all values of K when $n = 7$ within the set number of configurations

Analysis

15	✓	✓	✓	✓	✓	⊗	-	-
14	✓	✓	✓	✓	✓	⊗	-	-
13	✓	✓	✓	✓	✓	⊗	-	-
12	✓	✓	✓	✓	✓	⊗	-	-
11	✓	✓	✓	✓	✓	⊗	-	-
10	✓	✓	✓	✓	✓	⊗	-	-
9	✓	✓	✓	✓	✓	⊗	-	⊗
8	✓	✓	✓	✓	✓	⊗	⊗	⊗
7	✓	✓	✓	✓	✓	⊗	⊗	⊗
6	✓	✓	✓	✓	⊗	⊗	⊗	⊗
5	✓	✓	✓	⊗	⊗	⊗	⊗	⊗
4	✓	✓	⊗	⊗	⊗	⊗	⊗	⊗
3	✓	⊗	⊗	⊗	⊗	⊗	⊗	⊗
2	⊗	⊗	⊗	⊗	⊗	⊗	⊗	⊗
$K \backslash n$	3	4	5	6	7	8	9	10

- All behaviors for $n \leq 6$ were verified
- Showing convergence consistent with theoretically proven results
- The model exhibits non divergence for all values of K when $n = 7$ within the set number of configurations
- Divergence is demonstrated for all studied rings when $K \leq n - 1$.

Reducing Redundant Configurations: Offset-Equivalence



- Across the full benchmark, OE improves the PAR(1) score by 16.707% and solves 12 additional instances.
- For convergence, OE reduces PAR(1) by 11.01% improvement and solves 4 additional instances.
- For divergence, OE reduces PAR(1) by 31.78% improvement and solves 8 additional instances.

Restricted Daemon Assumptions

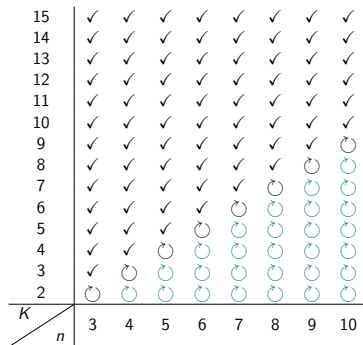
Results:

- There exists **synchronous** executions that **diverge** for any directed rooted ring of size n when $K = n - 1$.
- There exists **sequential** executions that **diverge** for any directed rooted ring of size n when $K < n - 1$.

Open Questions:

- (i) whether there exists **non-synchronous** executions that **diverge** for any directed rooted ring of size n when $K = n - 1$.
- (ii) whether there exists **non-sequential** executions that **diverge** for any directed rooted ring of size n when $K < n - 1$.

From Synchronous to Non-sequential Daemon



- Under a non-sequential daemon, the detected cycles can be synchronous.
- The K -state algorithm diverges for $K < n - 1$ under a **synchronous** daemon.
- The result also holds under a **non-sequential** daemon.

Non-synchronous Daemon

15	✓	✓	✓	✓	⊗	-	-	-
14	✓	✓	✓	✓	⊗	-	-	-
13	✓	✓	✓	✓	⊗	-	-	-
12	✓	✓	✓	✓	⊗	-	-	-
11	✓	✓	✓	✓	⊗	-	-	-
10	✓	✓	✓	✓	⊗	-	-	-
9	✓	✓	✓	✓	✓	-	-	-
8	✓	✓	✓	✓	⊗	-	-	⊗
7	✓	✓	✓	✓	✓	-	⊗	⊗
6	✓	✓	✓	✓	✓	⊗	⊗	⊗
5	✓	✓	✓	✓	⊗	⊗	⊗	⊗
4	✓	✓	✓	✓	⊗	⊗	⊗	⊗
3	✓	✓	✓	✓	⊗	⊗	⊗	⊗
2	✓	⊗	⊗	⊗	⊗	⊗	⊗	⊗
$K \backslash n$	3	4	5	6	7	8	9	10

- Only convergence occurs for ring sizes ranging from 3 to 7
- Solver-generated results already provide several counterexamples
- The K -state algorithm does not diverge on every directed rooted ring for $K = n - 1$ under a **non-synchronous** daemon.

Conclusion and Perspectives

- Formal SAT-based framework to analyze the behavior of Dijkstra's K -state algorithm under different daemon assumptions.
- Detection of convergence and divergence by encoding system states and executions as logical constraints.
- Deeper insight into the algorithm's behavior in asynchronous settings.
- Proposed offset-equivalence to reduce redundant configurations and improve solver performance.
- Apply the framework to other self-stabilizing algorithms.

On the Self-Stabilization of Dijkstra's Asynchronous Token Circulation

Asma Khoualdia, Sami Cherif, Stéphane Devismes, Léo Robert.

In CP 2026, 20-23 July 2026, Lisbon, Portugal

Thank you for your attention !

asma.khoualdia@u-picardie.fr