Self-stabilization Introduction

INFORMATIQUE
Sciences
Université Paris Cité

MPRI

Lélia Blin

lelia.blin@irif.fr
2023

Distributed system

Characterization of distributed systems

- Interconnected autonomous computing entities.
- Interaction through a means of communication.
- Lack of coordinator.

Goal of distributed system

Collaboration of the Entities to Achieve Common Tasks:

- Spanning structure
- Routing tables
- Mutual exclusion
- ...

Vocabulary

- Entities : processes or **nodes**,
- Interconnected : **network** organisation
- Global Task : **distributed algorithm** (or protocol)

Collaboration : Exchange of Informations

> A node can transmit to and get **information** from a part of other node.

Information exchanges are assumed **here** to be bidirectional: a node v can obtain information from node u if and only if u can obtain information from v.

Network Modelisation

Undirected graph $G = (V, E)$
-V is the set of nodes
- E is a set of edges

If the edge $\{v, u\}$ exists then v and u can directly exchange information together.

Graph terminology

We use the terminology of graph theory

- For example: For every edge $\{v, u\}$, v and u are said to be **neighbors**.
- Path, degree, diameter, tree, ...

Node Characterizations

- Each node has:
  - An **identifier**, or not
    - Anonymous network or identified
  - Its own **memory**
    - Non-corruptible memory (identifier, code, …)
    - Corruptible memory (variable contents)
  - Its own computational power
  - Its own clock

Distributed Algorithm

- Every node runs the same sequential algorithm denoted as $\mathcal{A}$
- The algorithm $\mathcal{A}$ features:
  - Traditional sequential algorithms with variables, tests, loops, etc.
  - Additional actions related to information reception
  - Triggers information sending

Local variables

- Each node run its algorithm $\mathcal{A}$
- Thus, each node v maintains its own variables of algorithm $\mathcal{A}$
- Global notation :
  - if the algorithm $\mathcal{A}$ has two variables dis and tmp
  - for the node v, these variables are denoted as dis_v and tmp_v
  - for the node u, these variables are denoted as dis_u and tmp_u
  - ....

Configurations

> **Node State**: It's the set containing the values of all its variables.

> For a graph $G(V, E)$, a **configuration** γ represents the states of all nodes at a specific time t.

The set of all the configurations are denoted by Γ.

Legitimate and Illegitimate Configurations

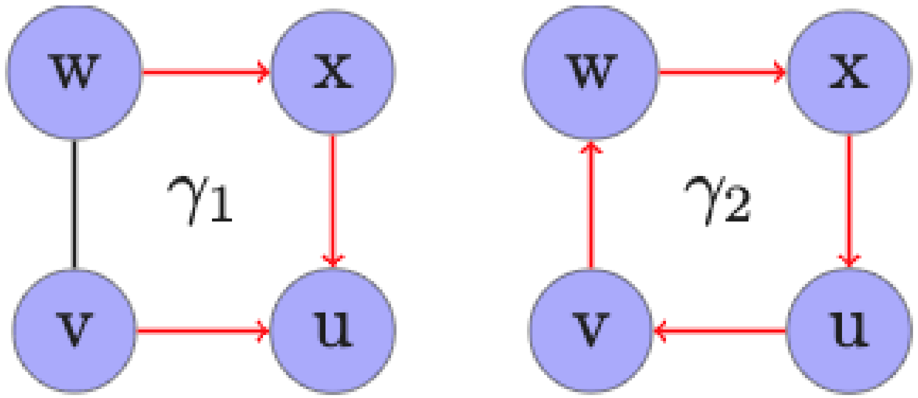
- Let $\mathcal{T}$ be the task to be solved by the distributed algorithm $\mathcal{A}$.

> - If configuration γ corresponds to $\mathcal{T}$, then γ is called a **legitimate** configuration; otherwise, it is termed **illegitimate**.

- Note: The terms "legal" and "illegal" are also used interchangeably with legitimate and illegitimate.

Example

- $\mathcal{T}$ Spanning Tree
- Local variable p for parent
- γ_1 legitimate configuration
- $\{p_u = \emptyset, p_v = u, p_w = x, p_x = u\}$
- $\gamma_2$illegitimate configuration
- $\{p_u = v, p_v = w, p_w = x, p_x = u\}$



Distributed Systems Nowadays

- Combinatorial explosion in the number of calculation entities
- Heterogeneity:
  - calculation entities
  - communications medium

Consequences

- Systems difficult to initialize
- Emergence of faults

Type of Faults

Generally, three types of possible failures are distinguished:

1. **Transient failures**: Failures of an arbitrary nature may strike the system, but there exists a point in the execution after which these failures no longer appear.
2. **Permanent failures**: Failures of an arbitrary nature may strike the system, but there exists a point in the execution after which these failures permanently incapacitate those affected by them.
3. **Intermittent failures**: Failures of an arbitrary nature may strike the system at any point during the execution.

Of course, transient and permanent failures are two specific cases of intermittent failures.

Fault tolerance

Robust algorithms

- **Leveraging Redundancy:** Utilize multiple layers of redundancy in:
  - Information
  - Communications
  - System nodes
- **Objective:** Ensure safe code execution through ample cross-checking and validation.
- **Underlying Assumption:** The system is designed to withstand a limited number of faults, always aiming to preserve a majority of correct elements, even in the face of more severe faults.
- **Characteristic:** Such algorithms are typically masking.

Self-Stabilizing Algorithms

- **Assumption:** Failures are transient (limited in time), but there's no constraint on the extent of the faults which might affect all system elements.
- **Definition:** An algorithm is considered self-stabilizing if it can reach legitimate configuration in finite time and stay in legitimate configurations, irrespective of the initial configuration.
- **Characteristic:** Typically non-masking. Between the cessation of faults and the system stabilization towards correct behavior, execution might be somewhat erratic.

Robust Algorithms

Pros:

- Intuitively align with fault-tolerance.
- Redundancy: Replace every element with three identical ones for better reliability.
- Actions are decided by majority consensus for reliable behavior.

Cons:

- Triple redundancy can lead to resource inefficiency and increased costs.
- May not handle arbitrary states resulting from faults as effectively as self-stabilizing algorithms.

Self-Stabilizing Algorithms

Pros:

- Tied to the concept of convergence; seeks a fixed point regardless of initial position.
- Can start from an arbitrary configuration.
- Capable of correct behavior within finite time, even if started in an unknown configuration.
- Used in many computer network protocols.

Cons:

- Operating from an arbitrary state can be counterintuitive in some contexts.
- Possible erratic behavior before achieving stabilization.

Concept of Self Stabilization

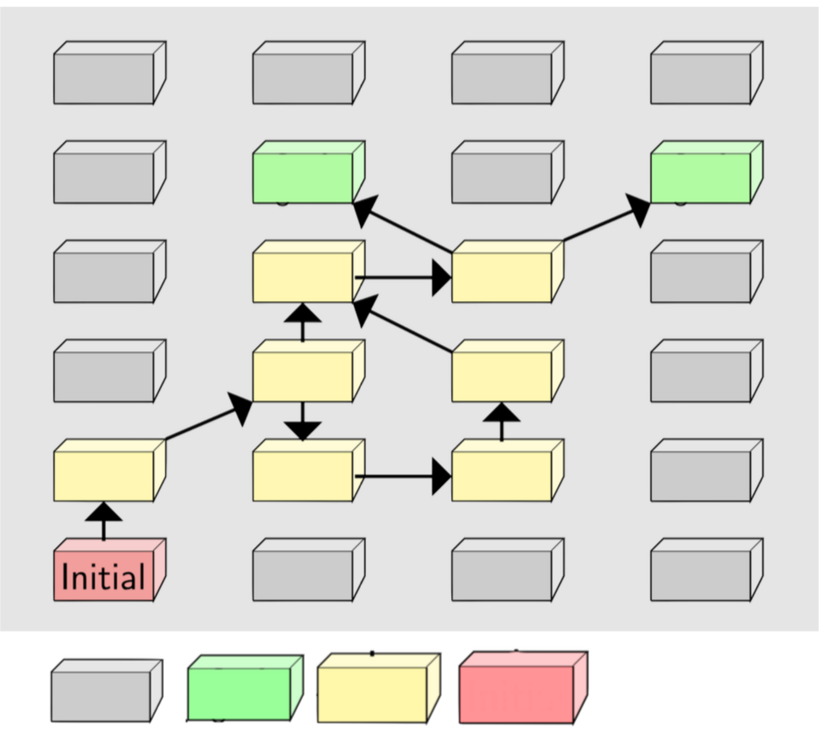
- Introduces by Dijkstra in 1974
- **Edsger Wybe Dijkstra**
    - 1930-2002
    - Computer scientist
    - Turing Award 1972
    - Dijkstra Prize

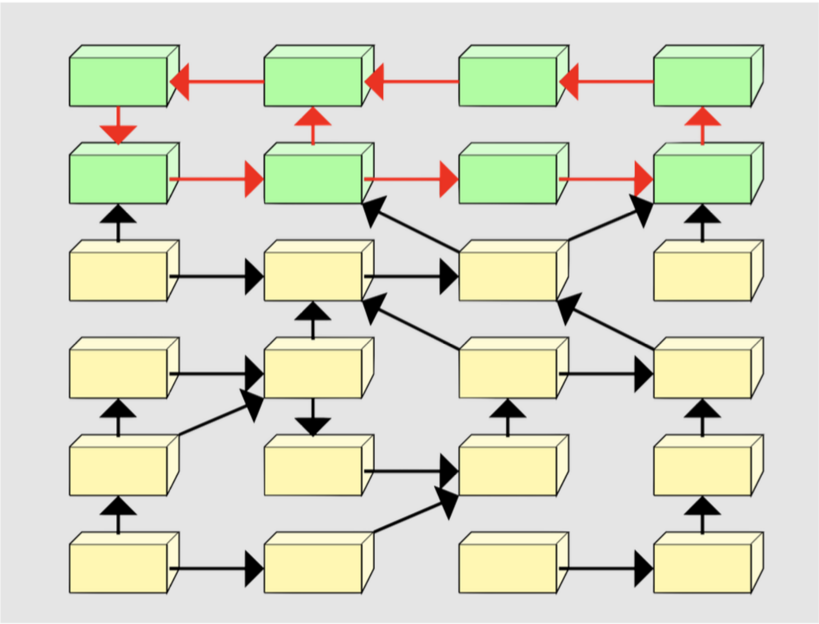
Definition

A self-stabilizing algorithm that solves task $\mathcal{T}$ is a distributed algorithm $\mathcal{A}$ satisfying:

1. **Convergence**: Starting from an arbitrary configuration, the system reaches a legitimate configuration using algorithm $\mathcal{A}$.
2. **Closure**: Starting from a legitimate configuration, the system remains in a legitimate configuration.

Classic algorithms vs Self-Stabilisation





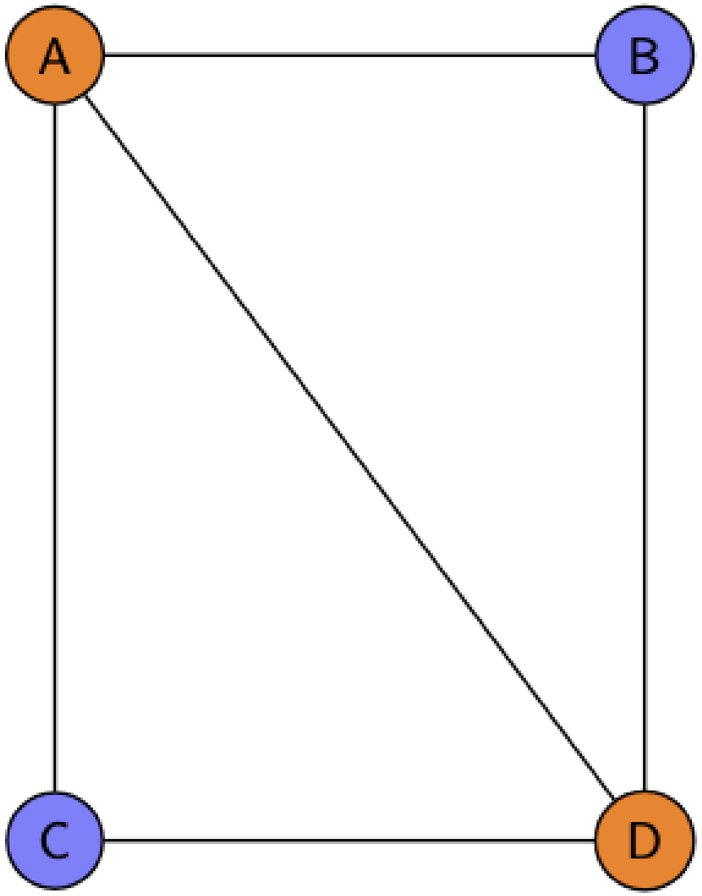
1. Locally detect illegal/legal configurations.
2. Return to a legal configuration.

Local detection of illegitimate configurations

A self-stabilizing algorithm must not only **construct a solution** but also provide a mechanism to **verify** that solution.

Local detection of illegitimate configurations

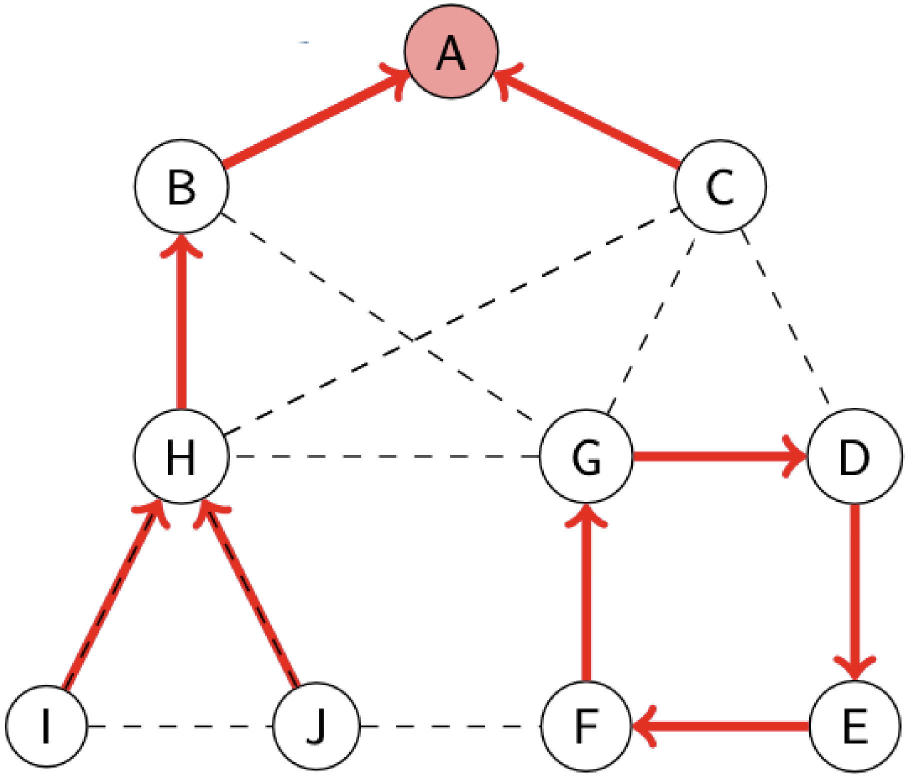
A self-stabilizing algorithm must not only **construct a solution** but also provide a mechanism to **verify** that solution.



Coloration

Local detection of illegitimate configurations

A self-stabilizing algorithm must not only **construct a solution** but also provide a mechanism to **verify** that solution.



Spanning Tree

Framework

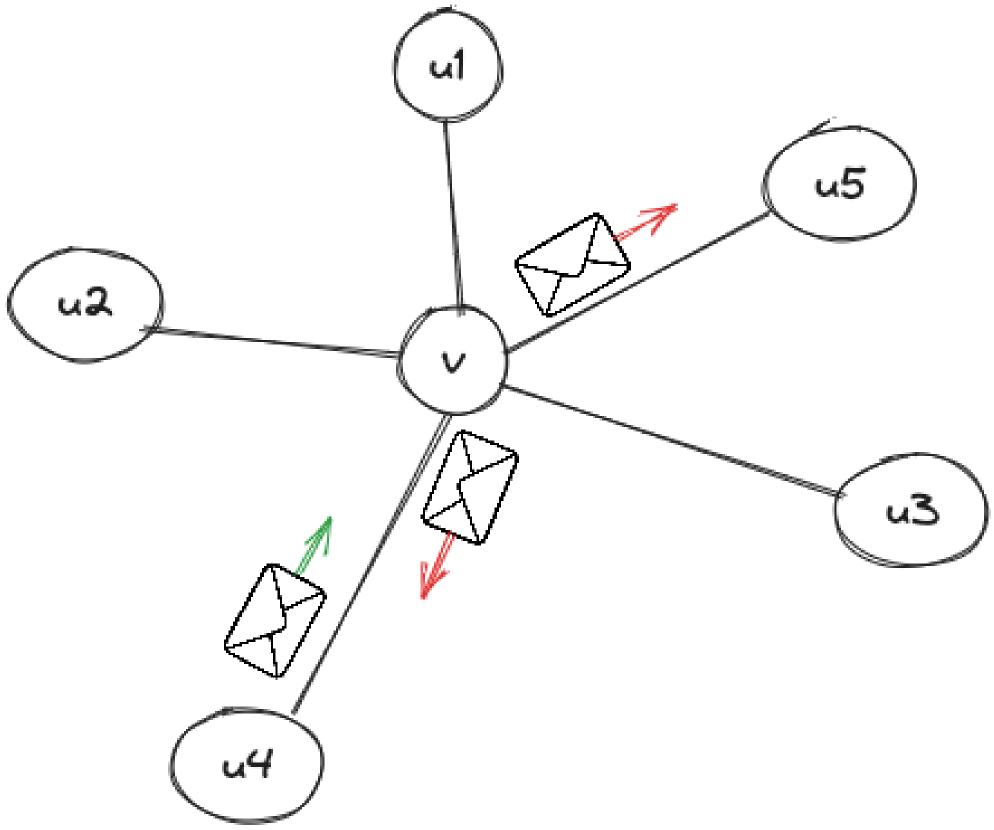
Computational model

3 main models of the litterature

Atomic step : fundamental unity

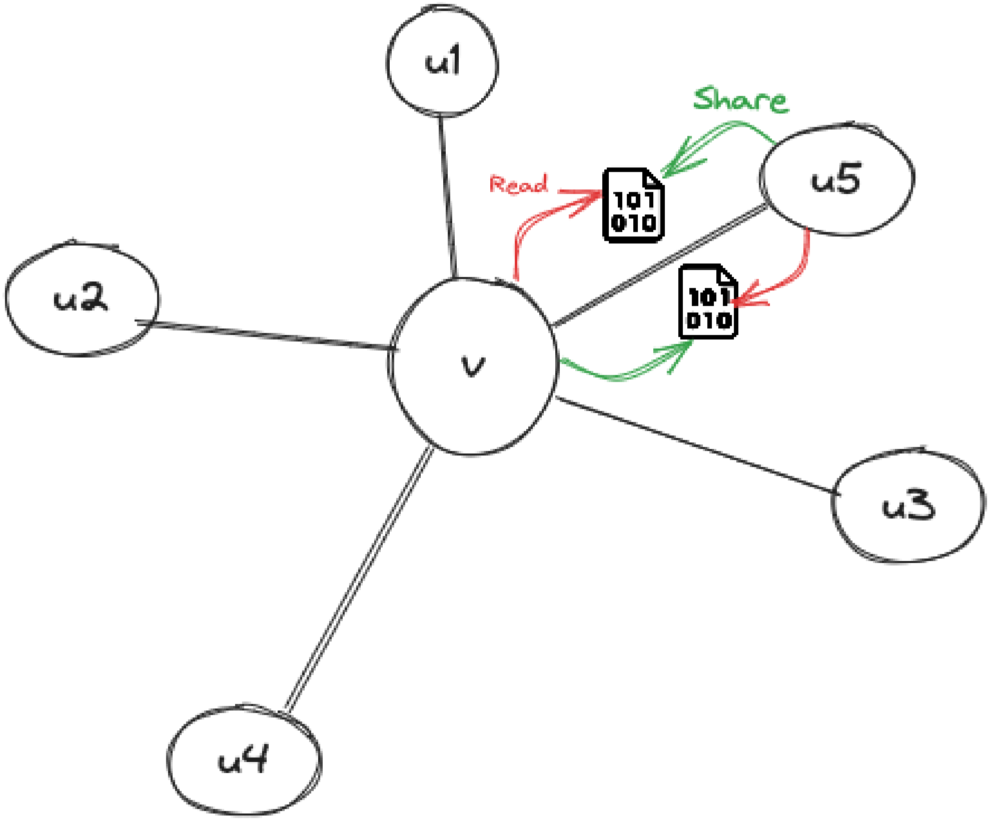
Message Passing

In one atomic step, a node either sends a message to one neighboring node or receives a message from one, but not both simultaneously.



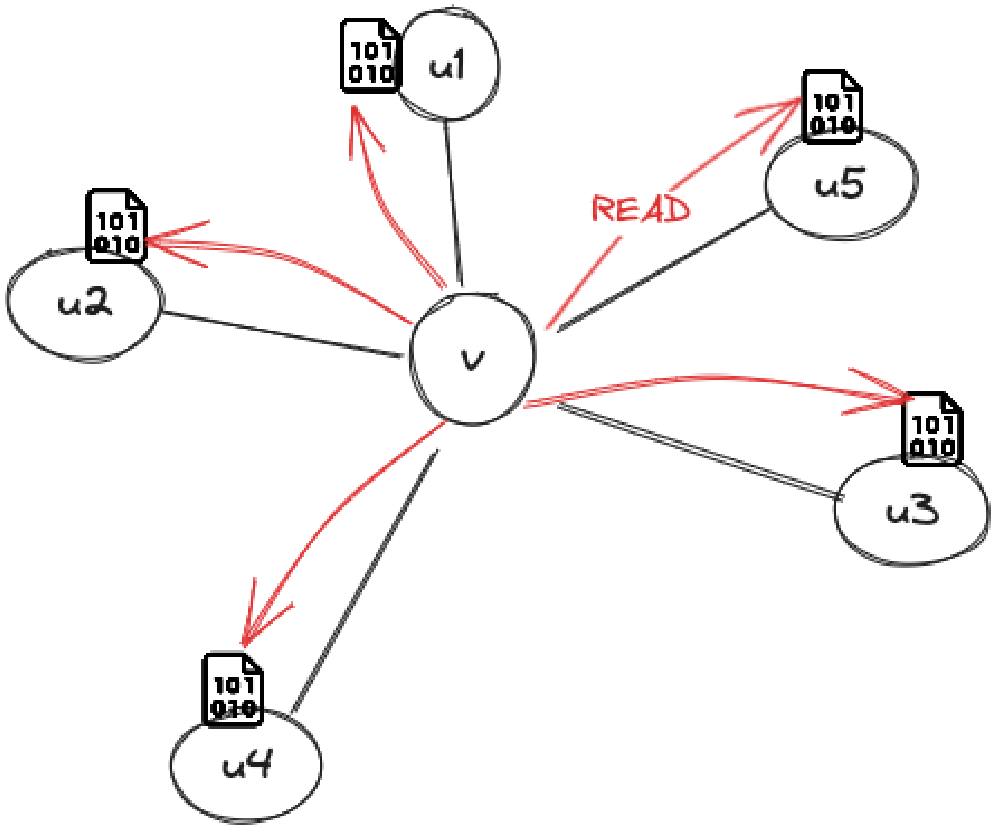
Shared Register

In one atomic step, a node can either read the state of one neighboring node or update its own state, but not both simultaneously.



Shared Memory

In one atomic step, a node can read the state of each neighboring node and update its own state.



Self-stabilizing algorithm

$$\langle label \rangle : \langle guard \rangle \longrightarrow \langle command \rangle$$

- **Labels** are only used to identify actions in the reasoning.
- A **guard** is a boolean predicate over node variables.
- A **command** is a set of variable-assignments.

Enabled nodes

- An action can be executed only if its guard evaluates to true;
- in this case, the action is said to be **enabled**.
- By extension, a node is said to be **enabled** if at least one of its actions is enabled.

Asynchronism

- Executions are driven by a nondeterministic adversary which models the asynchronism of the system.
- This adversary is called **daemon** or **scheduler**.

Computation Step

- Note that at any given time, while the scheduler can choose a subset of enabled nodes, it must pick at least one.
- After the atomic activation of all the enabled nodes chosen by the scheduler, a new configuration is obtained.

Fairness

Fairness allows to regulate the relative execution rate of processes by taking past actions into account. It is a liveness property in the sense of Alpern and Schneider [AS85].

Fairness assumptions

The three most popular fairness assumptions of the literature.

> A scheduler is
>
> - **strongly fair**, if it activates infinitely often all processes that are enabled infinitely often.
> - **weakly fair**, if it eventually activates every continuously enabled process.
> - **unfair** , has no fairness constraint
>   -it might never select a process unless it is the only enabled one.

Complexity

Time Complexity

The main units of measurement are used in the atomic-state model:

- The complexity in **rounds** evaluates the execution time according to the speed of the slowest processes
- The complexity in **moves** or **steps** captures the amount of computations an algorithm needs.

Space Complexity

First Self-Stabilizing Algorithm

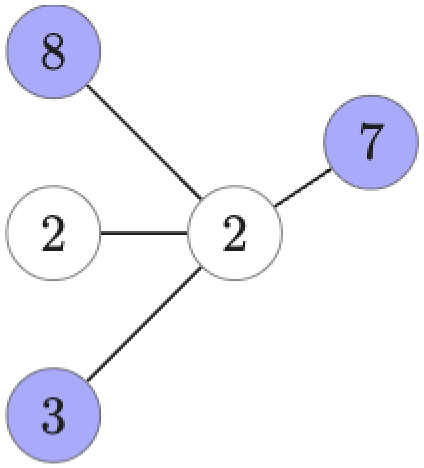
Example: Agreement on the same value

Agreement on the same value

Local variable: val_v is a positive integer $\forall v \in V$

Algorithm

$\exists u \in N(v))|val_u < val_v \to val_v := \min\{val_u | u \in N(v)\}$



Correctness

- Let denoted by m the minimun value $m = \min\{val_v : \forall v \in V\}$
- Let ψ :$\Gamma \times V \to \mathbb{N}$ be the function defined by:

$$\psi(\gamma, v) = val_v - m$$

- Let $\Psi : \Gamma \to \mathbb{N}$ be the function defined by:

$$\Psi(\gamma) = \sum_{v \in V} \psi(\gamma, v)$$

- And let defined by γ_{ell} the set of legal configurations
  - $\Gamma_\ell = \{\gamma \in \Gamma : \Psi(\gamma) = 0\}$

Theorem convergence: $true \triangleright \Gamma_\ell$

Proof: Let denoted by γ_0 an arbitrary illegal configuration, $\mathcal{E}(\gamma)$ the set of enabled node and $\mathcal{S}(\gamma)$ the set of enabled nodes choose by the scheduler.
$\forall v \in \mathcal{S}(\gamma)$ after execution of the algorithm by the node v we obtain $val_v(\gamma') < val_v(\gamma_0)$ so $\psi(\gamma', v) < \psi(\gamma_0, v)$ and $\Psi(\gamma) < \Psi(\gamma_0)$.

Theorem closure: Γ_ℓ is closed

Proof: Let $\gamma \in \Gamma_\ell$, so $\forall v \in V$ in γ we have $\psi(\gamma, v) = 0$ and $val_v = m$ as a consequence $\mathcal{E}(\gamma) = \emptyset$.

Time complexity

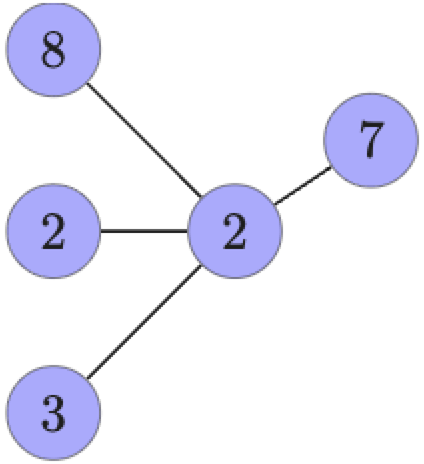
- $O(n)$ rounds (Which scheduler?)
- $0(n^2)$ steps (Which scheduler?)

Variant: Agreement on the same value

Local variable: val_v is a positive integer $\forall v \in V$

Algorithm

$\exists u \in N(v))|val_u \leq val_v \to val_v := \min\{val_u | u \in N(v)\}$



Silent property

> A self-stabilizing algorithm is termed "silent" if, once a legal configuration is reached, it remains in that same legal configuration.

Self-stabilization: Conclusion

Pros

- The network does not need to be initialized
- When a fault is diagnosed, it is sufficient to identify, then remove or restart the faulty components
- The self-stabilization property does not depend on the nature of the fault
- The self-stabilization property does not depend on the extent of the fault

Self-stabilization

Cons

- A priori, “eventually” does not give any bound on the stabilization time
- A priori, nodes never know whether the system is stabilized or not
- A single failure may trigger a correcting action at every node in the network
- Faults must be sufficiently rare that they can be considered are transient

Ressources

- **Self-Stabilization** , Shlomi Dolev MIT Press 2000
- **Introduction to Distributed Self-Stabilizing Algorithms**, Karine Altisen, Séphane Devisme, Swan Dubois, Franck Petit, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers 2019
- https://pages.lip6.fr/Franck.Petit/enseign/master/HCMV/RAD/lectures/Self-Stabilization_4pages.pdf