Acyclic Strategy for Silent Self-Stabilization in Spanning Forests

Karine Altisen, Stéphane Devismes, Anaïs Durand

To cite this version:

Karine Altisen, Stéphane Devismes, Anaïs Durand. Acyclic Strategy for Silent Self-Stabilization in Spanning Forests. SSS 2018 - 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems, Nov 2018, Tokyo, Japan. pp.186-202, 10.1007/978-3-030-03232-6_13 . hal-01938671

HAL Id: hal-01938671

https://hal.archives-ouvertes.fr/hal-01938671

Submitted on 28 Nov 2018

Acyclic Strategy for Silent Self-Stabilization in Spanning Forests[*]

Karine Altisen[1], Stéphane Devismes[1], and Anaïs Durand[2]

[1]Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000 Grenoble, France
[2]IRISA, Université de Rennes, 35042 Rennes, France

November 28, 2018

Abstract

We formalize design patterns, commonly used in self-stabilization, to obtain general statements regarding both correctness and time complexity. Precisely, we study a class of algorithms devoted to networks endowed with a sense of direction describing a spanning forest whose characterization is a simple (*i.e.*, quasi-syntactic) condition. We show that any algorithm of this class is (1) silent and self-stabilizing under the distributed unfair daemon, and (2) has a stabilization time polynomial in moves and asymptotically optimal in rounds. To illustrate the versatility of our method, we review several works where our results apply.

1 Introduction

Numerous self-stabilizing algorithms have been proposed so far to solve various tasks. Those works also consider a large taxonomy of topologies: rings [5], (directed) trees [9, 26], planar graphs [19], arbitrary connected graphs [1], *etc.* Among those topologies, the class of directed (in-)trees is of particular interest. Indeed, such topologies often appear, at an intermediate level, in self-stabilizing composite algorithms. *Composition* is a popular way to design self-stabilizing algorithms [25] since it allows to simplify both the design and the proofs. Numerous self-stabilizing algorithms [2, 4, 11] are actually made as a composition of a spanning directed treelike (*e.g.*, tree or forest) construction and some other algorithms specifically designed for directed tree/forest topologies. Notice that, even though not mandatory, most of these constructions additionally achieve *silence* [17]: a silent algorithm converges within finite time to a configuration from which the values of the communication registers used by the algorithm remain fixed. Silence is a desirable property, as it usually implies more simplicity in the design, and so allows to write simpler proofs; moreover, a silent algorithm may utilize fewer communication operations and communication bandwidth. We consider here the locally shared memory model with composite atomicity, where executions proceed in atomic steps and the asynchrony is captured by the notion of *daemon*. The most general daemon is the *distributed unfair daemon*. Hence, solutions stabilizing under such an assumption are highly desirable, because they work under any other daemon assumption. The daemon assumption and time complexity are closely related. The *stabilization time* (the main time complexity measure to compare self-stabilizing algorithms) is usually evaluated in terms of rounds, which capture the execution time according to the speed of the slowest processes. But, another crucial issue is the number of local state updates, called *moves*. Indeed, the stabilization time in moves captures the amount of computations an algorithm needs to recover a correct behavior. Now, this complexity can be bounded only if the algorithm works under an unfair daemon. If an algorithm requires

[*]This study has been partially supported by the ANR projects DESCARTES (ANR-16-CE40-0023) and ESTATE (ANR-16-CE25-0009), and by the Franco-German DFG-ANR project 40300781 DISCMAT.

a stronger daemon to stabilize, *e.g.*, a *weakly fair* daemon, then it is possible to construct executions whose convergence is arbitrarily long in terms of atomic steps (and so in moves), meaning that, in such executions, there are processes whose moves do not make the system progress towards the convergence. In other words, these latter processes waste computation power and so energy. Such a situation should be therefore prevented, making solutions working under the unfair daemon more desirable. There are many self-stabilizing algorithms proven under the distributed unfair daemon, *e.g.*, [1, 12, 20]. However, analyses of the stabilization time in moves is rather unusual and this may be an important issue. Indeed, recently, several self-stabilizing algorithms which work under a distributed unfair daemon have been shown to have an exponential stabilization time in moves in the worst case, *e.g.*, the silent leader election algorithms from [12] (see [1]), the Breadth-First Search (BFS) algorithm of Huang and Chen [21] (see [16]).

Contribution. We formalize design patterns, commonly used in self-stabilization, to obtain general statements regarding both correctness and time complexity. Precisely, we study a class of algorithms for networks endowed with a sense of direction describing a spanning forest (*e.g.*, a directed tree, or a network equipped with a spanning tree) whose characterization is a simple (*i.e.*, quasi-syntactic) condition. We show that any algorithm of this class is (1) silent and self-stabilizing under the distributed unfair daemon, and (2) has a stabilization time which is polynomial in moves and asymptotically optimal in rounds. Our condition mainly uses the concept of *acyclic strategy*, which is based on the notions of *top-down* and *bottom-up* actions. Our first goal has been to formally define these two paradigms. We have combined this formalization together with a notion of acyclic causality between actions and a last criteria called *correct-alone* (*n.b.*, only this criteria is not syntactic) to obtain the notion of *acyclic strategy*. We show that any algorithm following an acyclic strategy reaches a terminal configuration in a polynomial number of moves, assuming a distributed unfair daemon. Hence, if its terminal configurations satisfy the specification, the algorithm is both silent and self-stabilizing. Unfortunately, we show that this condition is not sufficient to obtain an asymptotically optimal stabilization time in rounds. So, we enforce the acyclic strategy with the property of *local mutual exclusivity* to have an asymptotically optimal round complexity. We also propose a simple method to make any algorithm, that follows an acyclic strategy, locally mutually exclusive. This method has no overhead in moves. Finally, to show the versatility of our approach, we review works where our results apply.

Related Work. General schemes and efficiency are usually understood as orthogonal issues. For example, the general scheme proposed in [23] transforms almost any algorithm working on an asynchronous message-passing identified system of arbitrary topology into its corresponding self-stabilizing version. Such a universal transformer is, by essence, inefficient in space and time complexities: its purpose is only to demonstrate the feasibility of the transformation. However, few works [18, 13, 3] target both general self-stabilizing algorithm patterns and efficiency in rounds. In [18, 13], authors propose a method to design silent self-stabilizing algorithms for a class of fix-point problems. Their solution works in non-bidirectional networks using bounded memory per process. In [18], they consider the locally shared memory model with composite atomicity assuming a distributed unfair daemon, while in [13], they bring their approach to asynchronous message-passing systems. In both papers, they establish a stabilization time in $O(D)$ rounds, where D is the network diameter, that holds for the synchronous case only. Moreover, move complexity is not considered. The rest of the related work only concerns the locally shared memory model with composite atomicity assuming a distributed unfair daemon. In [3], labeling schemes [24] are used to show that every static task has a silent self-stabilizing algorithm which converges within a linear number of rounds in an arbitrary identified network, however no move complexity is given. To our knowledge, until now, only two works [10, 15] conciliate general schemes for stabilization and efficiency in both moves and rounds. In [10], Cournier *et al* propose a general scheme for snap-stabilizing wave, henceforth non-silent, algorithms in arbitrary connected and rooted networks. Using their approach, one can obtain snap-stabilizing algorithms that execute each wave in polynomial number of rounds and moves. In [15], authors propose a general scheme to compute, in a linear number of rounds, spanning directed treelike data structures on arbitrary networks. They also show polynomial upper bounds on its stabilization time in moves holding for several instantiations of their scheme. Our approach is then complementary to [15].

Roadmap. In Section 2, we define the model. In Section 3, we define the *acyclic strategy* and propose a toy example. In Section 4, we study the move complexity of algorithms that follow an acyclic strategy. In Section 5, we analyze our case study regarding our results. In Section 6, we consider the round complexity issue. In Section 7, we review several existing works where our method applies. We conclude in Section 8.

2 Preliminaries

A *network* is made of a set of n interconnected *processes*. Communications are bidirectional. Hence, the topology of the network is a simple undirected graph $G = (V, E)$, where V is a set of processes and E is a set of edges that represents communication links, *i.e.*, $\{p, q\} \in E$ means that p and q can directly exchange information. In this latter case, p and q are said to be *neighbors*. For any process p, we denote by $p.\Gamma$ the set of its neighbors. We also note Δ the degree of G. A *distributed algorithm* $\mathcal{A}$ is a collection of $n = |V|$ *local algorithms*, each one operating on a single process: $\mathcal{A} = \{\mathcal{A}(p) \,:\, p \in V\}$ where each process p is equipped with a local algorithm $\mathcal{A}(p) = (Var_p, \mathit{Actions}_p)$: Var_p is the finite set of variables of p, and $\mathit{Actions}_p$ is the finite set of *actions*. Notice that $\mathcal{A}$ may not be uniform. We identify each variable involved in Algorithm $\mathcal{A}$ by the notation $p.x \in Var_p$, where x is the *name* of the variable and p the process that holds it. Each process p runs its local algorithm $\mathcal{A}(p)$ by atomically executing actions. If executed, an action of p consists of reading all variables of p and its neighbors, and then writing into a part of the *writable* variables of p. For any process p, each action in $\mathit{Actions}_p$ is written as follows: $L(p) \;::\; G(p) \;\to\; S(p)$. $L(p)$ is a *label* used to identify the action in the discussion. The *guard* $G(p)$ is a Boolean predicate involving variables of p and its neighbors. The *statement* $S(p)$ is a sequence of assignments on writable variables of p. A variable $q.x$ is said to be *G-read* by $L(p)$ if $q.x$ is involved in predicate $G(p)$ (in this case, q is either p or one of its neighbors). Let *G-Read*$(L(p))$ be the set of variables that are G-read by $L(p)$. A variable $p.x$ is said to be *written* by $L(p)$ if $p.x$ appears as the left operand in an assignment of $S(p)$. Let *Write*$(L(p))$ be the set of variables written by $L(p)$. An action can be executed by a process p only if it is *enabled*, *i.e.*, its guard evaluates to true. By extension, a process is *enabled* when at least one of its actions is enabled. The *state* of a process p is a vector of valuations of its variables. A *configuration* of an algorithm $\mathcal{A}$ is a vector made of a state of each process in V. For any configuration γ, we denote by $\gamma(p)$ (*resp.* $\gamma(p).x$) the state of process p (resp. the value of the variable x of process p) in γ.

The asynchrony of the system is modeled by the *daemon*. Assume that the current configuration of the system is γ. If the set of enabled processes in γ is empty, then γ is said to be *terminal*. Otherwise, a *step of* $\mathcal{A}$ is performed as follows: the daemon selects a non-empty subset S of enabled processes in γ, and every process p in S *atomically* executes the statement of one of its actions enabled in γ, leading the system to a new configuration γ'. The step (of $\mathcal{A}$) from γ to γ' is noted $\gamma \mapsto \gamma'$: $\mapsto$ is the binary relation over configurations defining all possible steps of $\mathcal{A}$ in G. An *execution* of $\mathcal{A}$ is a maximal sequence $\gamma_0\gamma_1...\gamma_i...$ of configurations such that $\gamma_{i-1} \mapsto \gamma_i$ for all $i > 0$. The term "maximal" means that the execution is either infinite, or ends at a *terminal* configuration. We define a daemon $\mathcal{D}$ as a predicate over executions. An execution e is then said to be an *execution under the daemon* $\mathcal{D}$ if e satisfies $\mathcal{D}$. Here, we assume that the daemon is *distributed* and *unfair*. "Distributed" means that, unless the configuration is terminal, the daemon selects at least one enabled process (maybe more) at each step. "Unfair" means that there is no fairness constraint, *i.e.*, the daemon might never select a process unless it is the only enabled one.

We measure the time complexity using two notions: *rounds* and *moves*. A process *moves* in $\gamma_i \mapsto \gamma_{i+1}$ when it executes an action in $\gamma_i \mapsto \gamma_{i+1}$. The definition of round uses the concept of *neutralization*:

a process v is *neutralized* during a step $\gamma_i \mapsto \gamma_{i+1}$, if v is enabled in γ_i but not in configuration γ_{i+1}, and it is not activated in the step $\gamma_i \mapsto \gamma_{i+1}$. The first round of an execution $e = \gamma_0\gamma_1...$ is its minimal prefix e' such that every process that is enabled in γ_0 either executes an action or is neutralized during a step of e'. If e' is finite, then the second round of e is the first round of the suffix $\gamma_t\gamma_{t+1}...$ of e starting from the last configuration γ_t of e', and so forth.

Let $\mathcal{A}$ be a distributed algorithm for a network G, SP a predicate over the configurations of $\mathcal{A}$, and $\mathcal{D}$ a daemon. $\mathcal{A}$ is *silent and self-stabilizing for SP in G under $\mathcal{D}$* if the following two conditions hold: (1) every execution of $\mathcal{A}$ under $\mathcal{D}$ is finite, and (2) every terminal configuration of $\mathcal{A}$ satisfies SP. In this case, every terminal (resp. non-terminal) configuration is said to be *legitimate w.r.t. SP* (resp. *illegitimate w.r.t. SP*). The *stabilization time* in rounds (resp. moves) of a silent self-stabilizing algorithm is the maximum number of rounds (resp. moves) over every execution possible under the considered daemon to reach a terminal (legitimate) configuration.

3 Algorithm with Acyclic Strategy

Let $\mathcal{A}$ be a distributed algorithm running on some network $G = (V, E)$.

Variable Names. We assume that every process is endowed with the same set of variables and we denote by $Names$ the set of names of those variables, namely: $Names = \{x : p \in V \wedge p.x \in Var_p\}$. We also assume that for every name $x \in Names$, for all processes p and q, variables $p.x$ and $q.x$ have the same definition domain. The set of names is partitioned into two subsets: $ConstNames$, the set of constant names, and $VarNames = Names \setminus ConstNames$, the set of writable variable names. A name x is in $VarNames$ as soon as there exists a process p such that $p.x \in Var_p$ and $p.x$ is written by an action of its local algorithm $\mathcal{A}(p)$. For every $c \in ConstNames$ and every process $p \in V$, $p.c$ is never written by any action and it has a pre-defined constant value (which may differ from one process to another, *e.g.*, Γ, the name of the neighborhood).

We assume that $\mathcal{A}$ is *well-formed*, *i.e.*, $VarNames$ is partitioned into k sets Var_1, ..., Var_k such that $\forall p \in V$, $\mathcal{A}(p)$ consists of exactly k actions $A_1(p), ..., A_k(p)$ where $\mathit{Write}(A_i(p)) = \{p.v : v \in Var_i\}$, for all $i \in \{1, ..., k\}$. Let $A_i = \{A_i(p) : p \in V\}$, for all $i \in \{1, ..., k\}$. Every A_i is called a *family (of actions)*. By definition, $A_1, ..., A_k$ is a partition over all actions of $\mathcal{A}$, henceforth called a *families' partition*.

Remark 1. *Since $\mathcal{A}$ is assumed to be* well-formed, *there is exactly one action of $\mathcal{A}(p)$ where $p.v$ is written, for every process p and every writable variable $p.v$ (of p).*

Spanning Forest. We assume that every process is endowed with constants that define a spanning forest over the graph G: we assume the constant names par and $chldrn$ such that for every process $p \in V$, $p.par$ and $p.chldrn$ are preset as follows.

- $p.par \in p.\Gamma \cup \{\bot\}$: $p.par$ is either a neighbor of p (its *parent* in the forest), or $\bot$. In this latter case, p is called a *(tree) root.* Hence, the graph made of vertices V and edges $\{(p, p.par) : p \in V \wedge p.par \neq \bot\}$ is assumed to be a spanning forest of G.
- $p.chldrn \subseteq p.\Gamma$: $p.chldrn$ contains the neighbors of p which are the *children* of p in the forest, *i.e.*, for every $p, q \in V$, $p.par = q \iff p \in q.chldrn$. If $p.chldrn = \emptyset$, p is called a *leaf.*

Notice that $p.\Gamma \setminus (\{p.par\} \cup p.chldrn)$ may not be empty. The set of p's *ancestors*, $Anc(p)$, is recursively defined as follows: $Anc(p) = \{p\}$ **if** p is a root, $Anc(p) = \{p\} \cup Anc(p.par)$ **otherwise**. Similarly, the set of p's *descendants*, $Desc(p)$, can be recursively defined as follows: $Desc(p) = \{p\}$ **if** p is a leaf, $Desc(p) = \{p\} \cup \bigcup_{q \in p.chldrn} Desc(q)$ **otherwise**.

Acyclic Strategy. Let $A_1, ..., A_k$ be the families' partition of $\mathcal{A}$. A_i, with $i \in \{1, ..., k\}$, is said to be *correct-alone* if for every process p and every step $\gamma \mapsto \gamma'$ such that $A_i(p)$ is executed in $\gamma \mapsto \gamma'$, if no variable in $\mathit{G\text{-}Read}(A_i(p)) \setminus \mathit{Write}(A_i(p))$ is modified in $\gamma \mapsto \gamma'$, then $A_i(p)$ is disabled in γ'. Notice that if a variable in *Write*$(A_i(p))$ is modified in $\gamma \mapsto \gamma'$, then it is necessarily modified by $A_i(p)$, by Remark 1.

Let $\prec_{\mathcal{A}}$ be a binary relation over the families of actions of $\mathcal{A}$ such that for $i, j \in \{1, ..., k\}$, $A_j \prec_{\mathcal{A}} A_i$ if and only if $i \neq j$ and there exist two processes p and q such that $q \in p.\Gamma \cup \{p\}$ and *Write*$(A_j(p)) \cap$ *G-Read*$(A_i(q)) \neq \emptyset$. We conveniently represent the relation $\prec_{\mathcal{A}}$ by a directed graph $\mathbf{GC}$ called *Graph of actions' Causality* and defined as follows: $\mathbf{GC} = (\{A_1, ..., A_k\}, \{(A_j, A_i), A_j \prec_{\mathcal{A}} A_i\})$.

Intuitively, a family of actions A_i is top-down if activations of its corresponding actions are only propagated down in the forest, *i.e.*, when some process q executes action $A_i(q)$, $A_i(q)$ can only activate A_i at some of its children p, if any. In this case, $A_i(q)$ writes to some variables G-read by $A_i(p)$, these latter are usually G-read to be compared to variables written by $A_i(p)$ itself. In other words, a variable G-read by $A_i(p)$ can be written by $A_i(q)$ only if $q = p$ or $q = p.par$. Formally, a family of actions A_i is *top-down* if for every process p and every $q.v \in$ *G-Read*$(A_i(p))$, we have $q.v \in$ *Write*$(A_i(q)) \Rightarrow q \in \{p, p.par\}$. Bottom-up families are defined similarly: a family A_i is *bottom-up* if for every process p and every $q.v \in$ *G-Read*$(A_i(p))$, we have $q.v \in$ *Write*$(A_i(q)) \Rightarrow q \in p.chldrn \cup \{p\}$.

A distributed algorithm $\mathcal{A}$ follows an *acyclic strategy* if it is well-formed, its graph of actions' causality $\mathbf{GC}$ is *(directed) acyclic*, and for every A_i in its families' partition, A_i is correct-alone and either bottom-up or top-down.

Toy Example. We now propose a simple example of an algorithm, called $\mathcal{TE}$, that follows an acyclic strategy. $\mathcal{TE}$ assumes a constant integer input $p.in \in \mathbb{N}$ at each process. $\mathcal{TE}$ computes the sum of all inputs and then spreads this result everywhere in the network. $\mathcal{TE}$ assumes that the network $T = (V, E)$ is a tree with a sense of direction (given by par and $chldrn$) which orientates T as an in-tree rooted at process r. Apart from those constant variables, every process p has two variables: $p.sub \in \mathbb{N}$ (which is used to compute the sum of input values in the subtree of p) and $p.res \in \mathbb{N}$ (which stabilizes to the result of the computation). $\mathcal{TE}$ consists of two families of actions S and R. S computes variables sub and is defined as follows. For every process p,

$$S(p) :: \ p.sub \neq (\sum_{q \in p.chldrn} q.sub) + p.in \to p.sub \leftarrow (\sum_{q \in p.chldrn} q.sub) + p.in$$

R computes variables res and is defined as follows.

$$R(r) :: r.res \neq r.sub \to r.res \leftarrow r.sub$$

For every process $p \neq r$,

$$R(p) :: p.res \neq \max(p.par.res, p.sub) \to p.res \leftarrow \max(p.par.res, p.sub)$$

S is bottom-up and correct-alone, while R is top-down and correct-alone. Moreover, the graph of actions' causality is simply $S \longrightarrow R$. So, $\mathcal{TE}$ follows an acyclic strategy.

4 Move Complexity of Algorithms with Acyclic Strategy

We now exhibit a polynomial upper bound on the move complexity of any algorithm that follows an acyclic strategy. To that goal, we consider a distributed algorithm $\mathcal{A}$ which follows an *acyclic strategy* and runs on the network $G = (V, E)$. We use the same notation as in Section 3, *e.g.*, we let $A_1, ..., A_k$ be the families' partition of $\mathcal{A}$.

For a process p and a family of actions A_i, $i \in \{1, ..., k\}$, we define the *impacting zone* of p and A_i, denoted $Z(p, A_i)$, as follows: $Z(p, A_i)$ is the set of p's ancestors **if** A_i is top-down, $Z(p, A_i)$ is the set of p's descendants **otherwise** (*i.e.*, A_i is bottom-up). Roughly speaking, a process q belongs to $Z(p, A_i)$ if the execution of $A_i(q)$ may cause an execution of $A_i(p)$ in the future.

Remark 2. *By definition, we have* $1 \leq |Z(p, A_i)| \leq n$. *Moreover, if* A_i *is top-down, then we have* $1 \leq |Z(p, A_i)| \leq H + 1 \leq n$, *where* H *is the height of* G, i.e., *the maximum among the heights trees of the forest.*

We also define the quantity $M(A_i,p)$ as the *level*[1] of p in G **if** A_i is top-down, the *height* of p in G **otherwise** (*i.e.*, A_i is bottom-up).

Remark 3. *By definition, we have* $0 \leq M(A_i,p) \leq H$*, where* H *is the height of* G*.*

We define $Others(A_i,p) = \{q \in p.\Gamma \ : \ \exists A_j, i \neq j \wedge \textit{Write}(A_j(q)) \cap \textit{G-Read}(A_i(p)) \neq \emptyset\}$ to be the set of neighbors q of p that have actions other than $A_i(q)$ which write variables that are G-read by $A_i(p)$. Let

$$maxO(A_i) = \max(\{|Others(A_i,p)| \ : \ p \in V\} \cup \{maxO(A_j) \ : \ A_j \prec_{\mathcal{A}} A_i)\})$$

Remark 4. *By definition, we have* $maxO(A_i) \leq \Delta$*. Moreover, if* $\forall p \in V$*,* $\forall i \in \{1,...,k\}$*,* $Others(A_i,p)$ *is empty, then* $\forall j \in \{1,...,k\}$*,* $maxO(A_j) = 0$*.*

Lemma 1. *Let* A_i *be a family of actions and* p *be a process. For every execution* e *of the algorithm* $\mathcal{A}$ *on* G*,* $\#m(e,A_i,p) \leq \Bigg(n \cdot \big(1 + \mathbf{d} \cdot \big(1 + maxO(A_i)\big)\big)\Bigg)^{\mathfrak{H}(A_i)} \cdot |Z(p,A_i)|$*, where* $\#m(e,A_i,p)$ *is the number of times* p *executes* $A_i(p)$ *in* e*,* $\mathbf{d}$ *is the in-degree of* $\mathbf{GC}$*, and* $\mathfrak{H}(A_i)$ *is the height of* A_i *in* $\mathbf{GC}$*.*[2]

Proof. Let $e = \gamma_0...\gamma_x...$ be any execution of $\mathcal{A}$ on G. Let $K(A_i,p) = M(A_i,p) + (H+1) \cdot \mathfrak{H}(A_i)$. We proceed by induction on $K(A_i,p)$.

Base Case: Assume $K(A_i,p) = 0$ for some family A_i and some process p. By definition, $H \geq 0$, $\mathfrak{H}(A_i) \geq 0$ and $M(A_i,p) \geq 0$. Hence, $K(A_i,p) = 0$ implies that $\mathfrak{H}(A_i) = 0$ and $M(A_i,p) = 0$. Since $M(A_i,p) = 0$, $Z(p,A_i) = \{p\}$. A_i is top-down or bottom-up so, for every $q.v \in \textit{G-Read}(A_i(p))$, $q.v \in \textit{Write}(A_i(q)) \Rightarrow q = p$. Moreover, since $\mathfrak{H}(A_i) = 0$, $\forall j \neq i$, $A_j \not\prec_{\mathcal{A}} A_i$. So, for every $j \neq i$ and every $q \in p.\Gamma \cup \{p\}$, $\textit{Write}(A_j(p)) \cap \textit{G-Read}(A_i(q)) = \emptyset$. Hence, no action except $A_i(p)$ can modify a variable in $\textit{G-Read}(A_i(p))$. Thus, $\#m(e,A_i,p) \leq 1$ since A_i is correct-alone.

Induction Hypothesis: Let $K \geq 0$. Assume that for every family A_j and every process q such that $K(A_j,q) \leq K$, we have

$$\#m(e,A_j,q) \leq \Bigg(n \cdot \big(1 + \mathbf{d} \cdot \big(1 + maxO(A_j)\big)\big)\Bigg)^{\mathfrak{H}(A_j)} \cdot |Z(q,A_j)|$$

Induction Step: Assume that for some family A_i and some process p, $K(A_i,p) = K+1$. If $\#m(e,A_i,p)$ equals 0 or 1, then the result trivially holds. Assume now that $\#m(e,A_i,p) > 1$ and consider two consecutive executions of $A_i(p)$ in e, *i.e.*, there exist x,y such that $0 \leq x < y$, $A_i(p)$ is executed in both $\gamma_x \mapsto \gamma_{x+1}$ and $\gamma_y \mapsto \gamma_{y+1}$, but not in steps $\gamma_z \mapsto \gamma_{z+1}$ with $z \in \{x+1,...,y-1\}$. Then, since A_i is correct-alone, at least one variable in $\textit{G-Read}(A_i(p))$ has to be modified by an action other than $A_i(p)$ in a step $\gamma_z \mapsto \gamma_{z+1}$ with $z \in \{x,...,y-1\}$ so that $A_i(p)$ becomes enabled again. Namely, there are $j \in \{1,...,k\}$ and $q \in V$ such that *(a)* $j \neq i$ *or* $q \neq p$, $A_j(q)$ is executed in a step $\gamma_z \mapsto \gamma_{z+1}$, and $\textit{Write}(A_j(q)) \cap \textit{G-Read}(A_i(p)) \neq \emptyset$. Note also that, by definition, *(b)* $q \in p.\Gamma \cup \{p\}$. Finally, by definitions of top-down and bottom-up, *(a)*, and *(b)*, $A_j(q)$ satisfies: (1) $j \neq i \wedge q = p$, (2) $j = i \wedge q \in p.\Gamma \cap Z(p,A_i)$, or (3) $j \neq i \wedge q \in p.\Gamma$. In other words, at least one of the three following cases occurs:

(1) *p executes $A_j(p)$ in step $\gamma_z \mapsto \gamma_{z+1}$ with $j \neq i$ and $\textit{Write}(A_j(p)) \cap \textit{G-Read}(A_i(p)) \neq \emptyset$.* Consequently, $A_j \prec_{\mathcal{A}} A_i$ and, so, $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$. Moreover, $M(A_j,p) - M(A_i,p) \leq H$ and $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$ imply $K(A_j,p) < K(A_i,p) = K+1$. Hence, by induction hypothesis, we have $\#m(e,A_j,p) \leq \Bigg(n \cdot \big(1 + \mathbf{d} \cdot \big(1 + maxO(A_j)\big)\big)\Bigg)^{\mathfrak{H}(A_j)} \cdot |Z(p,A_j)|$.

(2) *There is $q \in p.\Gamma \cap Z(p,A_i)$ such that q executes $A_i(q)$ in step $\gamma_z \mapsto \gamma_{z+1}$ and $\textit{Write}(A_i(q)) \cap \textit{G-Read}(A_i(p)) \neq \emptyset$.* Then, $M(A_i,q) < M(A_i,p)$. Since $M(A_i,q) < M(A_i,p)$, $K(A_i,q) <$

[1]The level of p in G is the distance from p to the root of its tree in G (0 if p is the root itself).

[2]The height of A_i in $\mathbf{GC}$ is 0 if the in-degree of A_i in $\mathbf{GC}$ is 0. Otherwise, it is equal to one plus the maximum of the heights of the A_i's predecessors *w.r.t.* $\prec_{\mathcal{A}}$.

$K(A_i,p) = K+1$ and, by induction hypothesis, we have
$\#m(e,A_i,q) \leq \Big(n \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)\Big)^{\mathfrak{H}(A_i)} \cdot |Z(q,A_i)|$.

(3) *A neighbor q of p executes an action $A_j(q)$ in step $\gamma_z \mapsto \gamma_{z+1}$, with $j \neq i$ and Write$(A_j(q)) \cap$ G-Read$(A_i(p)) \neq \emptyset$.* Consequently, $A_j \prec_{\mathcal{A}} A_i$ and, so, $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$. Moreover, $M(A_j,q) - M(A_i,p) \leq H$ and $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$ imply $K(A_j,q) < K(A_i,p) = K+1$. Hence, by induction hypothesis, we have
$\#m(e,A_j,q) \leq \Big(n \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_j)\big)\big)\Big)^{\mathfrak{H}(A_j)} \cdot |Z(q,A_j)|$.

(Notice that Cases 1 and 3 can only occur when $\mathfrak{H}(A_i) > 0$.) We now bound the number of times each of the three above cases occur in the execution e.

Case 1: By definition, there exist at most $\mathbf{d}$ predecessors A_j of A_i in $\mathbf{GC}$ (*i.e.*, such that $A_j \prec_{\mathcal{A}} A_i$). For each of them, we have $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$, $|Z(p,A_j)| \leq n$ (Remark 2) and $maxO(A_j) \leq maxO(A_i)$. Hence, overall, Case 1 appears at most $m_1 = \sum_{\{A_j \,:\, A_j \prec_{\mathcal{A}} A_i\}} \#m(e,A_j,p)$ times and

$$\begin{aligned} m_1 &\leq \sum_{\{A_j \,:\, A_j \prec_{\mathcal{A}} A_i\}} \Big(n \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_j)\big)\big)\Big)^{\mathfrak{H}(A_j)} \cdot |Z(p,A_j)| \\ &\leq \mathbf{d} \cdot n^{\mathfrak{H}(A_i)} \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)^{\mathfrak{H}(A_i)-1} \end{aligned}$$

Case 2: By definition, $Z(p,A_i) = \{p\} \uplus \biguplus_{q \in p.\Gamma \cap Z(p,A_i)} Z(q,A_i)$ Hence, overall, this case appears at most $m_2 = \sum_{q \in p.\Gamma \cap Z(p,A_i)} \#m(e,A_i,q)$ times and

$$\begin{aligned} m_2 &\leq \sum_{q \in p.\Gamma \cap Z(p,A_i)} \Big(n \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)\Big)^{\mathfrak{H}(A_i)} \cdot |Z(q,A_i)| \\ &\leq n^{\mathfrak{H}(A_i)} \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)^{\mathfrak{H}(A_i)} \cdot \big(|Z(p,A_i)|-1\big) \end{aligned}$$

Case 3: $q \in Others(A_i,p)$ since $i \neq j$ and $q \in p.\Gamma$. Then, for every $A_j \prec_{\mathcal{A}} A_i$, we have $\mathfrak{H}(A_j) < \mathfrak{H}(A_i)$, $maxO(A_j) \leq maxO(A_i)$, and $Z(q,A_j) \leq n$ (Remark 2). By definition, there are at most $\mathbf{d}$ families A_j such that $A_j \prec_{\mathcal{A}} A_i$. Finally, $|Others(A_i,p)| \leq maxO(A_i)$, by definition. Hence, overall, this case appears at most $m_3 = \sum_{\{A_j \,:\, A_j \prec_{\mathcal{A}} A_i\}} \sum_{\{q \in Others(A_i,p)\}} \#m(e,A_j,q)$ times and

$$\begin{aligned} m_3 &\leq \sum_{\{A_j \,:\, A_j \prec_{\mathcal{A}} A_i\}} \sum_{\{q \in Others(A_i,p)\}} \Big(n \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_j)\big)\big)\Big)^{\mathfrak{H}(A_j)} \cdot |Z(q,A_j)| \\ &\leq \mathbf{d} \cdot maxO(A_i) \cdot n^{\mathfrak{H}(A_i)} \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)^{\mathfrak{H}(A_i)-1} \end{aligned}$$

Hence, overall, we have

$$\begin{aligned} \#m(e,A_i,p) &\leq 1+m_1+m_2+m_3 \\ &\leq n^{\mathfrak{H}(A_i)} \cdot \big(1+\mathbf{d}\cdot\big(1+maxO(A_i)\big)\big)^{\mathfrak{H}(A_i)} \cdot |Z(p,A_i)| \end{aligned}$$

□

Since $maxO(A_i) \leq \Delta$ (Remark 4) and $|Z(p,A_i)| \leq n$ (Remark 2), we can deduce the following theorem from Lemma 1 and the definition of silent self-stabilization.

Theorem 1. *If $\mathcal{A}$ follows an* acyclic strategy *and every terminal configuration of $\mathcal{A}$ satisfies SP, then (1) $\mathcal{A}$ is silent and self-stabilizing for SP in G under the distributed unfair daemon, and (2) its stabilization time is at most $\big(1+\mathbf{d}\cdot(1+\Delta)\big)^{\mathfrak{H}} \cdot k \cdot n^{\mathfrak{H}+2}$ moves, where k is the number of families of $\mathcal{A}$, $\mathbf{d}$ is the in-degree of $\mathbf{GC}$, and $\mathfrak{H}$ the height of $\mathbf{GC}$.*

5 Analysis of $\mathcal{TE}$

We now analyze $\mathcal{TE}$ using our results. The aim is to show that: (1) correctness and move complexity of $\mathcal{TE}$ can be easily deduced from our general results, (2) our upper bound on stabilization time in moves is tight for this example, and (3) our definition of acyclic strategy does not preclude the design of solutions (like $\mathcal{TE}$) that are inefficient in terms of rounds. We will see how to circumvent this latter negative result in Section 6.

First, we already saw that $\mathcal{TE}$ follows an acyclic strategy and that the graph of actions' causality is simply $S \longrightarrow R$. Then, by induction on the tree T, we can show that every terminal configuration of $\mathcal{TE}$ is legitimate. Hence, by Theorem 1, we can conclude that $\mathcal{TE}$ is silent and self-stabilizing for computing the sum of the inputs assuming a distributed unfair daemon. Moreover, its stabilization time is at most $2 \cdot (2 + \Delta) \cdot n^3$ moves. Now, using Lemma 1, the move complexity of $\mathcal{TE}$ can be further refined. Let e be any execution and H be the height of T. First, note that $maxO(S) = maxO(R) = 0$ by Remark 4. Since S is bottom-up, $|Z(p, S)| \leq n$, for every process p. Moreover, the height of S is 0 in the graph of actions' causality. Hence, by Lemma 1, we have $\#m(e, S, p) \leq n$, for all processes p. Thus, e contains at most n^2 moves of S. Similarly, since R is top-down, $|Z(p, R)| \leq H + 1$, for every process p. Moreover, the height of R is 1 in the graph of actions' causality. Hence, by Lemma 1, we have $\#m(e, R, p) \leq 2 \cdot n \cdot (H + 1)$, for all processes p. Thus, e contains at most $2 \cdot n^2 \cdot (H + 1)$ moves of R. Overall, the stabilization time of $\mathcal{TE}$ is actually at most $n^2(3 + 2H)$ moves.

Lower Bound in Moves. We now show that the stabilization time of $\mathcal{TE}$ is $\Omega(H \cdot n^2)$ moves, meaning that the previous upper bound (obtained by Lemma 1) is asymptotically reachable. To that goal, we consider a directed line of n processes, with $n \geq 4$, noted $p_1, ..., p_n$: p_1 is the root and for every $i \in \{2, ..., n\}$, there is a link between p_{i-1} and p_i, moreover, $p_i.par = p_{i-1}$ (note that $H = n$). We build a possible execution of $\mathcal{TE}$ running on this line that contains $\Omega(H \cdot n^2)$ moves. We assume a central unfair daemon: at each step exactly one process executes an action. In this execution, we fix that $p_i.in = 1$, for every $i \in \{1, ..., n\}$. We consider two classes of configurations: Configurations X_{2i+1}(with $3 \leq 2i + 1 \leq n$) and Configurations Y_{2i+2} (with $4 \leq 2i + 2 \leq n$), see Figure 1. The initial configuration of the execution is X_3. Then, we proceed as follows: the system converges from configuration X_{2i+1} to configuration Y_{2i+2} in $\Omega(i^2)$ moves using Schedule 1 and then from Y_{2i+2} to X_{2i+3} in $\Omega(i)$ moves using Schedule 2, back and forth, until reaching a terminal configuration (X_n if n is odd, Y_n otherwise).

Configuration X_{2i+1}*,* $3 \leq 2i + 1 \leq n$*:*

	p_1	$\cdots$	p_{2i-2}	p_{2i-1}	p_{2i}	p_{2i+1}	p_{2i+2}	p_{2i+3}	p_{2i+4}	p_{2i+5}	$\cdots$
in	1	$\ldots$	1	1	1	1	1	1	1	1	$\ldots$
sub	$2i$	$\ldots$	3	2	1	0	$2i$	0	$2i+2$	0	$\ldots$
res	$2i$	$\ldots$	$2i$	$2i$	$2i$	0	0	0	0	0	$\ldots$

Configuration Y_{2i+2}*,* $4 \leq 2i + 2 \leq n$*:*

	p_1	$\cdots$	p_{2i-2}	p_{2i-1}	p_{2i}	p_{2i+1}	p_{2i+2}	p_{2i+3}	p_{2i+4}	p_{2i+5}	$\cdots$
in	1	$\ldots$	1	1	1	1	1	1	1	1	$\ldots$
sub	$4i+1$	$\ldots$	$2i+4$	$2i+3$	$2i+2$	$2i+1$	$2i$	0	$2i+2$	0	$\ldots$
res	$4i+1$	$\ldots$	$4i+1$	$4i+1$	$4i+1$	$4i+1$	0	0	0	0	$\ldots$

Figure 1: Configurations X_{2i+1} and Y_{2i+2}

Hence, following this scheduling of actions, the execution that starts in configuration X_3 converges to X_n (resp. Y_n) if n is odd (resp. even) and contains $\Omega(n^3)$ moves, precisely, $\Omega(H \cdot n^2)$ since the network is a line ($H = n - 1$).

Remark that in this execution, for every process p, when $R(p)$ is activated, $S(p)$ is disabled: this means that if the algorithm is modified so that $S(p)$ has local priority over $R(p)$ for every process p (like in the method proposed in Section 6), the proposed execution is still possible keeping a move complexity in $\Omega(H \cdot n^2)$ even for such a prioritized algorithm.

Schedule 1 from X_{2i+1} to Y_{2i+2}

```
1: for $j = 2i+1$ down to $1$ do
2:     $p_j$ executes $S(p_j)$
3:     for $k = j$ to $2i+1$ do
4:         $p_k$ executes $R(p_k)$
```

Schedule 2 from Y_{2i+2} to X_{2i+3}

```
1: for $j = 2i+2$ down to $1$ do
2:     $p_j$ executes $S(p_j)$
3: for $j = 1$ to $2i+1$ do
4:     $p_j$ executes $R(p_j)$
```

Lower Bound in Rounds. We now show that $\mathcal{TE}$ has a stabilization time in $\Omega(n)$ rounds in any tree of height $H = 1$, *i.e.*, a star network. This negative result is due to the fact that families R and S are not locally mutually exclusive. In the next section, we will propose a transformation to obtain a stabilization time in $O(H)$ rounds, so $O(1)$ rounds in the case of a star network, without affecting the move complexity.

We now construct a possible execution that terminates in $n+2$ rounds in a star network of n processes ($n \geq 2$): p_1 is the root of the tree and $p_2, ..., p_n$ are the leaves (namely links are $\{\{p_1, p_i\}, i = 2, ..., n\}$). We note C_i, $i \in \{1, ..., n\}$, the configuration satisfying the following three conditions:
(1) $\forall i \in \{1, ..., n\}$, $p_i.in = 1$;
(2) $p_1.sub = i$, $\forall j \in \{2, ..., i\}$, $p_j.sub = 1$, and $\forall j \in \{i+1, ..., n\}$, $p_j.sub = 0$; (3) $\forall i \in \{1, ..., n\}$, $p_i.res = i$.

In every configuration C_i, processes p_1, ..., p_i are disabled and processes $p_{i+1}, ..., p_n$ are enabled for S. We now build a possible execution that starts from C_1 and successively converges to configurations C_2, ..., C_n (C_n is a terminal configuration). To converge from C_i to C_{i+1}, $i \in \{1, ..., n-1\}$, the daemon applies Schedule 3. The convergence from C_1 to C_{n-1} last $n-2$ rounds since for $i \in \{1, ..., n-2\}$, the convergence from C_i to C_{i+1} lasts exactly one round. Indeed, each process executes at least one action between C_i and C_{i+1} and process p_n is enabled in configuration C_i and remains continuously enabled until being activated as the last process to execute in the round. The convergence from C_{n-1} to C_n lasts 4 rounds: in C_{n-1}, only p_n is enabled to execute $S(p_n)$ hence the round terminates in one step where only $S(p_n)$ is executed. Similarly, p_1 then sequentially executes $S(p_1)$ and $R(p_1)$ in two rounds. Finally, $p_2, ..., p_n$ execute R in one round and then the system is in the terminal configuration C_n. Hence the execution lasts $n+2$ rounds.

Schedule 3 from C_i to C_{i+1}

```
1: $p_{i+1}$ executes $S(p_{i+1})$
2: $p_1$ executes $S(p_1)$
3: $p_1$ executes $R(p_1)$
4: for $j = 2$ to $n$ do
5:     $p_j$ executes $R(p_j)$
```

6 Round Complexity of Algorithms with Acyclic Strategy

We now propose an extra condition that is sufficient for any algorithm following an acyclic strategy to stabilize in $O(H)$ rounds. We then propose a simple method to add this property to any algorithm that follows an acyclic strategy, without affecting the move complexity. Throughout this section, we consider a distributed well-formed algorithm $\mathcal{A}$ designed for a network G endowed with a spanning forest. Let $A_1, ..., A_k$ be the families' partition of $\mathcal{A}$.

A Condition for a Stabilization Time in $O(H)$ rounds. We say that families A_i and A_j are *locally mutually exclusive* if for every process p, there is no configuration γ where both $A_i(p)$ and $A_j(p)$ are enabled. By extension, we say $\mathcal{A}$ is *locally mutually exclusive* if $\forall i, j \in \{1, ..., k\}$, $i \neq j$ implies that A_i and A_j are *locally mutually exclusive*. Below, note that $\mathfrak{H} < k$ and, in usual cases, the number of families k is a constant.

Theorem 2. *If $\mathcal{A}$ follows an acyclic strategy and is locally mutually exclusive, then every execution of $\mathcal{A}$ reaches a terminal configuration within at most $(\mathfrak{H}+1) \cdot (H+1)$ rounds, where $\mathfrak{H}$ is the height of the graph of actions' causality of $\mathcal{A}$ and H is the height of the spanning forest.*

Proof Outline. Let A_i be a family of actions of $\mathcal{A}$ and p be a process. We note $R(A_i, p) = \mathfrak{H}(A_i) \cdot$

$(H+1)+M(A_i,p)+1$. We first show, by induction, that for every family A_i and every process p, after $R(A_i,p)$ rounds, $A_i(p)$ is disabled forever. Then, since for every family A_i and every process p, $\mathfrak{H}(A_i) \leq \mathfrak{H}$ and $M(A_i,p) \leq H$, we have $R(A_i,p) \leq (\mathfrak{H}+1)\cdot(H+1)$, and the theorem holds. □

A Transformer. We know that there exist algorithms that follow an acyclic strategy, are not locally mutually exclusive, and stabilize in $\Omega(n)$ rounds (see Section 5). We now formalize a method, based on priorities over actions, to transform such algorithms into locally mutually exclusive ones. This ensures a complexity in $O(H)$ rounds, without degrading the move complexity.

In the following, for every process p and every family A_i, we identify the guard and the statement of Action $A_i(p)$ by $G_i(p)$ and $S_i(p)$, respectively. Let $\lhd_{\mathcal{A}}$ be any strict total order on families of $\mathcal{A}$ compatible with $\prec_{\mathcal{A}}$, *i.e.*, $\lhd_{\mathcal{A}}$ is a binary relation on families of $\mathcal{A}$ that satisfies:
Strict Order: $\lhd_{\mathcal{A}}$ is irreflexive and transitive;
Total: for every two families A_i, A_j, we have either $A_i \lhd_{\mathcal{A}} A_j$, $A_j \lhd_{\mathcal{A}} A_i$, or $i=j$; and
Compatibility: for every two families A_i, A_j, if $A_i \prec_{\mathcal{A}} A_j$, then $A_i \lhd_{\mathcal{A}} A_j$.
Let $\mathtt{T}(\mathcal{A})$ be the following algorithm:
(1) $\mathtt{T}(\mathcal{A})$ and $\mathcal{A}$ have the same set of variables.
(2) Every process $p \in V$ holds k actions (recall that k is the number of families of $\mathcal{A}$): for every $i \in \{1,...,k\}$, $A^{\mathtt{T}}_i(p) :: G^{\mathtt{T}}_i(p) \rightarrow S^{\mathtt{T}}_i(p)$
where $G^{\mathtt{T}}_i(p) = \left(\bigwedge_{A_j \lhd_{\mathcal{A}} A_i} \neg G_j(p)\right) \wedge G_i(p)$ and $S^{\mathtt{T}}_i(p) = S_i(p)$.

$G_i(p)$ (resp. the set $\{G_j(p) : A_j \lhd_{\mathcal{A}} A_i\}$) is called the *positive part* (resp. *negative part*) of $G^{\mathtt{T}}_i(p)$. By definition, $\prec_{\mathcal{A}}$ is irreflexive and the graph of actions' causality induced by $\prec_{\mathcal{A}}$ is acyclic. So, there always exists a strict total order compatible with $\prec_{\mathcal{A}}$, *i.e.*, the above transformation is always possible for any algorithm $\mathcal{A}$ which follows an acyclic strategy. Moreover, by construction, we have the two following remarks:

Remark 5. *(1) $\mathtt{T}(\mathcal{A})$ is well-formed, (2) $A^{\mathtt{T}}_1,...,A^{\mathtt{T}}_k$ is the families' partition of $\mathtt{T}(\mathcal{A})$, where $A^{\mathtt{T}}_i = \{A^{\mathtt{T}}_i(p) : p \in V\}$, for every $i \in \{1,...,k\}$, and (3) $\mathtt{T}(\mathcal{A})$ is locally mutually exclusive.*

Remark 6. *For every $i,j \in \{1,...,k\}$ such that $i \neq j$, and every process p, the positive part of $G^{\mathtt{T}}_j(p)$ belongs to the negative part in $G^{\mathtt{T}}_i(p)$ if and only if $A_j \lhd_{\mathcal{A}} A_i$.*

Lemma 2. *For every $i,j \in \{1,...,k\}$, if $A^{\mathtt{T}}_j \prec_{\mathtt{T}(\mathcal{A})} A^{\mathtt{T}}_i$, then $A_j \lhd_{\mathcal{A}} A_i$.*

Proof. Let $A^{\mathtt{T}}_i$ and $A^{\mathtt{T}}_j$ be two families such that $A^{\mathtt{T}}_j \prec_{\mathtt{T}(\mathcal{A})} A^{\mathtt{T}}_i$. Then, $i \neq j$ and there exist two processes p and q such that $q \in p.\Gamma \cup \{p\}$ and $\mathit{Write}(A^{\mathtt{T}}_j(p)) \cap \mathit{G\text{-}Read}(A^{\mathtt{T}}_i(q)) \neq \emptyset$. Then, $\mathit{Write}(A^{\mathtt{T}}_j(p)) = \mathit{Write}(A_j(p))$, and either $\mathit{Write}(A_j(p)) \cap \mathit{G\text{-}Read}(A_i(q)) \neq \emptyset$, or $\mathit{Write}(A_j(p)) \cap \mathit{G\text{-}Read}(A_\ell(q)) \neq \emptyset$ where $G_\ell(q)$ belongs to the negative part of $G^{\mathtt{T}}_i(q)$. In the former case, we have $A_j \prec_{\mathcal{A}} A_i$, which implies that $A_j \lhd_{\mathcal{A}} A_i$ ($\lhd_{\mathcal{A}}$ is compatible with $\prec_{\mathcal{A}}$). In the latter case, $A_j \prec_{\mathcal{A}} A_\ell$ (by definition) and $A_\ell \lhd_{\mathcal{A}} A_i$ (by Remark 6). Since, $A_j \prec_{\mathcal{A}} A_\ell$ implies $A_j \lhd_{\mathcal{A}} A_\ell$ ($\lhd_{\mathcal{A}}$ is compatible with $\prec_{\mathcal{A}}$), by transitivity we have $A_j \lhd_{\mathcal{A}} A_i$. Hence, for every $i,j \in \{1,...,k\}$, $A^{\mathtt{T}}_j \prec_{\mathtt{T}(\mathcal{A})} A^{\mathtt{T}}_i$ implies $A_j \lhd_{\mathcal{A}} A_i$, and we are done.
□

Lemma 3. $\mathtt{T}(\mathcal{A})$ *follows an acyclic strategy.*

Proof. Let $A^{\mathtt{T}}_i$ be a family of $\mathtt{T}(\mathcal{A})$. The lemma is proven by the following three claims.
(1) $A^{\mathtt{T}}_i$ *is correct-alone.* Indeed, as A_i is correct-alone and for every process p, $S^{\mathtt{T}}_i(p) = S_i(p)$ and $\neg G_i(p) \Rightarrow \neg G^{\mathtt{T}}_i(p)$, we have that $A^{\mathtt{T}}_i$ is also correct-alone.
(2) $A^{\mathtt{T}}_i$ *is either bottom-up or top-down.* Since $\mathcal{A}$ follows an acyclic strategy, A_i is either bottom-up or top-down. Assume A_i is bottom-up. By construction, for every process q, $S^{\mathtt{T}}_i(q) = S_i(q)$ so $\mathit{Write}(A^{\mathtt{T}}_i(q)) = \mathit{Write}(A_i(q))$. Let $q.v \in \mathit{G\text{-}Read}(A^{\mathtt{T}}_i(p))$.

- Assume $q.v \in \mathit{G\text{-}Read}(A_i(p))$. Then $q.v \in \mathit{Write}(A_i(q)) \Rightarrow q \in p.\mathit{chldrn} \cup \{p\}$ (since A_i is bottom-up), *i.e.*, $q.v \in \mathit{Write}(A^{\mathtt{T}}_i(q)) \Rightarrow q \in p.\mathit{chldrn} \cup \{p\}$.

- Assume that $q.v \notin \textit{G-Read}(A_i(p))$. Then $q.v \in \textit{G-Read}(A_j(p))$ such that $G_j(p)$ belongs to the negative part of $G_i^{\mathtt{T}}(p)$, *i.e.*, $A_j \lhd_{\mathcal{A}} A_i$ (Remark 6). Assume, by the contradiction, that $q.v \in \textit{Write}(A_i^{\mathtt{T}}(q))$. Then $q.v \in \textit{Write}(A_i(q))$, and since $p \in q.\Gamma \cup \{q\}$ (indeed, $q.v \in \textit{G-Read}(A_j(p))$), we have $A_i \prec_{\mathcal{A}} A_j$. Now, as $\lhd_{\mathcal{A}}$ is compatible with $\prec_{\mathcal{A}}$, we have $A_i \lhd_{\mathcal{A}} A_j$. Hence, $A_j \lhd_{\mathcal{A}} A_i$ and $A_i \lhd_{\mathcal{A}} A_j$, a contradiction. Thus, $q.v \notin \textit{Write}(A_i^{\mathtt{T}}(q))$ which implies that $q.v \in \textit{Write}(A_i^{\mathtt{T}}(q)) \Rightarrow q \in p.chldrn \cup \{p\}$ holds in this case.

Hence, $A_i^{\mathtt{T}}$ is bottom-up. By a similar reasoning, if A_i is top-down, $A_i^{\mathtt{T}}$ is top-down too.
(3) *The graph of actions' causality of* $\mathtt{T}(\mathcal{A})$ *is acyclic.* Indeed, by Lemma 2, for every $i, j \in \{1, ..., k\}$, $A_j^{\mathtt{T}} \prec_{\mathtt{T}(\mathcal{A})} A_i^{\mathtt{T}} \Rightarrow A_j \lhd_{\mathcal{A}} A_i$. Now, $\lhd_{\mathcal{A}}$ is a strict total order. So, the graph of actions' causality of $\mathtt{T}(\mathcal{A})$ is acyclic. □

Lemma 4. *Every execution of* $\mathtt{T}(\mathcal{A})$ *is an execution of* $\mathcal{A}$.

Proof. $\mathcal{A}$ and $\mathtt{T}(\mathcal{A})$ have the same set of configurations; every step of $\mathtt{T}(\mathcal{A})$ is a step of $\mathcal{A}$; and a configuration γ is terminal *w.r.t.* $\mathtt{T}(\mathcal{A})$ iff γ is terminal *w.r.t.* $\mathcal{A}$. □

Theorem 3. *If* $\mathcal{A}$ *follows an acyclic strategy, and is silent and self-stabilizing for* SP *in* G *under the distributed unfair daemon, then*
(1) $\mathtt{T}(\mathcal{A})$ *is silent and self-stabilizing for* SP *in* G *under the distributed unfair daemon,*
(2) its stabilization time is at most $(\mathfrak{H} + 1) \cdot (H + 1)$ *rounds, and*
(3) its stabilization time in moves is less than or equal to the one of $\mathcal{A}$.
where $\mathfrak{H}$ *is the height of the graph of actions' causality of* $\mathcal{A}$ *and* H *is the height of the spanning forest.*

Proof. By Remark 5, Lemmas 3 and 4, and Theorems 1 and 2. □

Using the above theorem, we can apply the transformer on our toy example: $\mathtt{T}(\mathcal{TE})$ stabilizes in at most $2(H + 1)$ rounds and $\Theta(H \cdot n^2)$ moves in the worst case.

7 Related Work and Applications

There are many works [26, 8, 9, 6, 7, 22, 14] where we can apply our generic results. These works propose silent self-stabilizing algorithms for directed trees or network where a directed spanning tree is available. These algorithms are, or can be easily translated into, well-formed algorithms that follow an acyclic strategy. Hence, their correctness and time complexities (in moves and rounds) are directly deduced from our results. Below, we only present a few of them.

Turau and Köhler [26] proposes three algorithms for directed trees. Each algorithm is given with its proof of correctness and round complexity, however move complexity is not considered. These three algorithms can be trivially translated in our model, and our results allow to obtain the same round complexities, and additionally provide move complexities. Among those three algorithms, the third one is the most interesting since it uses 5 families of actions, while the two first use 1 and 2 families respectively. So, we only detail this latter. The algorithm computes a minimum connected distance-k dominating set using the five families below:

$$\begin{array}{llll}
A_1(p) & :: & p.L \neq \mathcal{L}(p) & \rightarrow \quad p.L \leftarrow \mathcal{L}(p) \\
A_2(p) & :: & p.level \neq level(p) & \rightarrow \quad p.level \leftarrow level(p) \\
A_3(p) & :: & p.cds \neq cds'(p) & \rightarrow \quad p.cds \leftarrow cds'(p) \\
A_4(p) & :: & p.cds \wedge p.dist_l \neq dist_l(p) & \rightarrow \quad p.dist_l \leftarrow dist_l(p) \\
A_5(p) & :: & p.minc \neq minc(p) & \rightarrow \quad p.minc \leftarrow minc(p)
\end{array}$$

We do not explain here the role of the variables nor their computation using macros, please refer to the original paper [26]. But from their definition in [26], we can observe that: (1) $\mathcal{L}(p)$ depends on $q.L$ for $q \in p.chldrn$; (2) $level(p)$ depends on $p.par.level$; (3) $cds'(p)$ that depend on $p.L$ and $q.cds$

for $q \in p.chldrn$; (4) $dist_l(p)$ depends on $q.L$ and $q.cds$ for $q \in p.chldrn$, and $p.par.dist_l$; finally (5) $minc(p)$ depends on $p.level$, $q.cds$ and $q.minc$ for $q \in p.chldrn$. So, A_1, A_3, A_5 are bottom-up and correct-alone and A_2, A_4 are top-down and correct-alone. The graph of actions' causality is acyclic since $A_1 \prec A_3$, $A_1 \prec A_4$, $A_2 \prec A_5$, $A_3 \prec A_4$, and $A_3 \prec A_5$; and its height is $\mathfrak{H} = 2$. Thus, as in [26], we have a round complexity in $O(H)$. Moreover, by Theorem 1, the move complexity is in $O(\Delta^2.n^4)$, where Δ is the degree of the tree.

The silent algorithm given in [22] finds articulation points in a network endowed with a breadth-first spanning tree, assuming a central unfair daemon. The algorithm computes for each node p the variable $p.e$ which contains every non-tree edges incident on p and some non-tree edges incident on descendants of p once a terminal configuration is reached. Precisely, a non-tree edge $\{p, q\}$ is propagated up in the tree starting from p and q until the first common ancestor of p and q. Based on $p.e$, the node p can decide whether or not it is an articulation point. The algorithm can be translated as a single family of actions which is correct-alone and bottom-up. So, it follows that this algorithm is silent and self-stabilizing even assuming a distributed unfair daemon. Moreover, its stabilization time is in $O(n^2)$ moves and $O(H)$ rounds.

The algorithm in [14] computes cut-nodes and bridges on connected graph endowed with a depth-first spanning tree. It is silent and self-stabilizing under a distributed unfair distributed daemon and converges within $O(n^2)$ moves and $O(H)$ rounds. Indeed, the algorithm contains a single family of actions which is correct-alone and bottom-up.

8 Conclusion

We have presented a general scheme to prove and analyze silent self-stabilizing algorithms executing on networks endowed with a sense of direction describing a spanning forest. Our results allow to easily (*i.e.* quasi-syntactically) deduce correctness and upper bounds on both move and round complexities of such algorithms. We have identified a number of algorithms [26, 8, 9, 6, 7, 22, 14] where our method applies. In several of those works, the assumption about the existence of a directed spanning tree has to be considered as an intermediate assumption, since this structure has to be built by an underlying algorithm. Now, several silent self-stabilizing spanning tree constructions are efficient in both rounds and moves, *e.g.*, [15]. Thus, both algorithms, *i.e.*, the one that builds the tree and the one that computes on this tree, have to be carefully composed to obtain a general composite algorithm where the stabilization time is kept both asymptotically optimal in rounds and polynomial in moves.

References

[1] K. Altisen, A. Cournier, S. Devismes, A. Durand, and F. Petit. Self-stabilizing leader election in polynomial steps. *Inf. Comput.*, 254:330–366, 2017.

[2] A. Arora, M. Gouda, and T. Herman. Composite routing protocols. In *SPDP'90*, pages 70–78, 1990.

[3] L. Blin, P. Fraigniaud, and B. Patt-Shamir. On proof-labeling schemes versus silent self-stabilizing algorithms. In *SSS'14*, pages 18–32, 2014.

[4] L. Blin, M. Potop-Butucaru, S. Rovedakis, and S. Tixeuil. Loop-free super-stabilizing spanning tree construction. In *SSS'10*, pages 50–64, 2010.

[5] L. Blin and S. Tixeuil. Compact deterministic self-stabilizing leader election on a ring: the exponential advantage of being talkative. *Dist. Comp.*, 31(2):139–166, 2018.

[6] P. Chaudhuri. An $O(n^2)$ Self-Stabilizing Algorithm for Computing Bridge-Connected Components. *Computing*, 62(1):55–67, 1999.

[7] P. Chaudhuri. A note on self-stabilizing articulation point detection. *Journal of Systems Architecture*, 45(14):1249–1252, 1999.

[8] P. Chaudhuri and H. Thompson. Self-stabilizing tree ranking. *Int. J. Comput. Math.*, 82(5):529–539, 2005.

[9] P. Chaudhuri and H. Thompson. Improved self-stabilizing algorithms for l(2, 1)-labeling tree networks. *Mathematics in Computer Science*, 5(1):27–39, 2011.

[10] A. Cournier, S. Devismes, and V. Villain. Light enabling snap-stabilization of fundamental protocols. *TAAS*, 4(1):6:1–6:27, 2009.

[11] A. K. Datta, S. Devismes, K. Heurtefeux, L. L. Larmore, and Y. Rivierre. Competitive self-stabilizing k-clustering. *TCS*, 626:110–133, 2016.

[12] A. K. Datta, L. L. Larmore, and P. Vemula. An $O(N)$-time self-stabilizing leader election algorithm. *JPDC*, 71(11):1532–1544, 2011.

[13] S. Delaët, B. Ducourthial, and S. Tixeuil. Self-stabilization with r-operators revisited. *JACIC*, 3(10):498–514, 2006.

[14] S. Devismes. A silent self-stabilizing algorithm for finding cut-nodes and bridges. *Parallel Processing Letters*, 15(1-2):183–198, 2005.

[15] S. Devismes, D. Ilcinkas, and C. Johnen. Silent self-stabilizing scheme for spanning-tree-like constructions. Technical report, HAL, Feb 2018.

[16] S. Devismes and C. Johnen. Silent self-stabilizing BFS tree algorithms revisited. *JPDC*, 97:11 – 23, 2016.

[17] S. Dolev, M. G. Gouda, and M. Schneider. Memory requirements for silent stabilization. *Acta Inf.*, 36(6):447–462, 1999.

[18] B. Ducourthial and S. Tixeuil. Self-stabilization with r-operators. *Dist. Comp.*, 14(3):147–162, 2001.

[19] S. Ghosh and M. H. Karaata. A self-stabilizing algorithm for coloring planar graphs. *Dist. Comp.*, 7(1):55–59, 1993.

[20] C. Glacet, N. Hanusse, D. Ilcinkas, and C. Johnen. Disconnected components detection and rooted shortest-path tree maintenance in networks. In *SSS'14*, pages 120–134, 2014.

[21] S.-T. Huang and N.-S. Chen. A self-stabilizing algorithm for constructing breadth-first trees. *IPL*, 41(2):109–117, 1992.

[22] M. H. Karaata. A self-stabilizing algorithm for finding articulation points. *Int. J. Found. Comput. Sci.*, 10(1):33–46, 1999.

[23] S. Katz and K. J. Perry. Self-stabilizing extensions for message-passing systems. *Dist. Comp.*, 7(1):17–26, 1993.

[24] A. Korman, S. Kutten, and D. Peleg. Proof labeling schemes. *Dist. Comp.*, 22(4):215–233, 2010.

[25] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, Cambridge, UK, Second edition 2001.

[26] V. Turau and S. Köhler. A distributed algorithm for minimum distance-k domination in trees. *J. Graph Algorithms Appl.*, 19(1):223–242, 2015.