Correctness of self-stabilizing algorithms under the Dolev model when adapted to composite atomicity models

Chih-Yuan Chen¹, Cheng-Pin Wang², Tetz C. Huang³ and Ji-Cherng Lin³

¹ Department of Computer Science and Information Engineering, Tao-yuan Innovation Institute of Technology, 414, Sec. 3, Chung-Shan East Road, Chung-Li, Tao-Yuan 320, Taiwan
² General Education Center, Tzu Chi College of Technology, 880, Sec. 2, Chien-kuo Rd., Hualien, Taiwan
³ Department of Computer Science and Engineering, Yuan-Ze University, 135 Yuan-Tung Road, Chung-Li, Tao-Yuan 320, Taiwan

Abstract

In this paper, we first clarify that it is not a trivial matter whether or not a self-stabilizing algorithm under the Dolev model, when adapted to a composite atomicity model, is also self-stabilizing. Then we employ a particular “simulation” approach to show that if a self-stabilizing algorithm under the Dolev model has one of two certain forms, then it is also self-stabilizing when adapted to one of the composite atomicity models, the fair daemon model. Since most existing self-stabilizing algorithms under the Dolev model have the above-mentioned forms, our results imply that they are all self-stabilizing when adapted to the fair daemon model.

Keywords: Silent self-stabilizing algorithm; Composite atomicity; Read/write atomicity; Fair daemon model; Adaptation of algorithm

1 Introduction

A distributed system consists of a set of loosely connected processors that do not share a global memory. It is usually modelled by a connected simple undirected graph \( G = (V, E) \), with each node \( x \in V \) representing a processor in the system and each edge \( \{x, y\} \in E \) representing the link connecting processors \( x \) and \( y \). Each processor has one or more shared registers and possibly some non-shared local variables, the contents of which specify the local state of the processor. Local states of all processors in the system at a certain time constitute the global configuration (or, simply, configuration) of the system at that time. The main restriction of the distributed system is that each processor in the system can only access the data (i.e., read the shared data) of its neighbors. Since a distributed algorithm is an algorithm that works in a distributed system, it cannot violate this main restriction. In this paper,
we adopt the point of view in (Dolev et al., 1993). Thus, an atomic step is the “largest” step that is guaranteed to be executed uninterruptedly. A distributed algorithm uses composite atomicity if some atomic step contains (at least) a read operation and a write operation. A distributed algorithm uses read/write atomicity if each atomic step contains either a single read operation or a single write operation but not both.

1.1 Computational models

1.1.1 Composite atomicity models

The Dijkstra’s central daemon model (or, simply, central daemon model) was first introduced in (Dijkstra, 1974). Under this computational model, each processor is equipped with a local algorithm that consists of one or more rules of the form:

\[ \text{condition part} \rightarrow \text{action part}. \]

The condition part (or guard) is a Boolean expression of registers of the processor and its neighbors, and the action part is an assignment of values to some registers of the processor. If the condition part of one or more rules in a processor is evaluated to be true, we say that the processor is privileged to execute the action part of any of these rules (or privileged to make a move). Under this computational model, if the system starts with a configuration in which no processor in the system is privileged, then the system is deadlocked. Otherwise, the central daemon in the system will randomly select exactly one privileged processor and exactly one executable rule in the processor’s local algorithm, and let the selected processor execute the action part of the selected rule. The local state of the selected processor thus changes, which in the meantime results in the change of the global configuration of the system. The system will repeat the above process to change configurations as long as it does not encounter any deadlock situation. Thus the behavior of the system under the action of the algorithm can be described by executions defined as follows: an infinite sequence of configurations \( \Gamma = (\gamma_1, \gamma_2, \ldots) \) of a distributed system is called an infinite execution (of the algorithm in the system) under the central daemon model if for any \( i \geq 1 \), \( \gamma_{i+1} \) is obtained from \( \gamma_i \) after exactly one processor in the system makes a move in the \( i^{th} \) step \( \gamma_i \rightarrow \gamma_{i+1} \); a finite sequence of configurations \( \Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k) \) of a distributed system is called a finite execution (of the algorithm in the system) under the central daemon model if (1) \( k = 1 \), or for any \( i = 1, 2, \ldots, k-1 \), \( \gamma_{i+1} \) is obtained from \( \gamma_i \) after exactly one processor in the system makes a move in the \( i^{th} \) step \( \gamma_i \rightarrow \gamma_{i+1} \), and (2) no node is privileged in the last configuration \( \gamma_k \).

The distributed daemon model was later considered in (Burns, 1987). The difference between the central daemon model and the distributed daemon model is the number of processors that make moves in a step of an execution of the algorithm. Under the central daemon model, exactly one privileged processor in the system is randomly selected by the central daemon to make a move in a step of an execution of the algorithm. Under the distributed daemon model, however, an arbitrary number of privileged processors are randomly selected by the distributed daemon to simultaneously make moves in a step. Thus, we can also define executions for the distributed daemon model as follows: an infinite sequence of configurations \( \Gamma = (\gamma_1, \gamma_2, \ldots) \) of a distributed system is called an infinite execution (of the algorithm in the system) under the distributed daemon model if for any \( i \geq 1 \), \( \gamma_{i+1} \) is obtained from \( \gamma_i \) after a certain number of privileged processors selected by the distributed daemon simultaneously make moves in the \( i^{th} \) step \( \gamma_i \rightarrow \gamma_{i+1} \); a finite sequence of configurations \( \Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_k) \) of a distributed system is called a finite execution (of the algorithm in the system) under the distributed daemon model if (1) \( k = 1 \), or for any \( i = 1, 2, \ldots, k-1 \), \( \gamma_{i+1} \) is obtained from \( \gamma_i \) after a certain number of privileged processors selected by the distributed daemon simultaneously make moves in the
$i^{th}$ step $\gamma_i \rightarrow \gamma_{i+1}$, and (2) no node is privileged in the last configuration $\gamma_k$. As a consequence of the above definitions, the central daemon model is a restricted version of the distributed daemon model, i.e., the set of all distributed-daemon-model executions contains the set of all central-daemon-model executions.

The weakly fair daemon model (or, simply, fair daemon model) is a different restricted version of the distributed daemon model. Precisely, an execution $\Gamma$ under the distributed daemon model is called an execution under the fair daemon model if for any suffix $\Gamma'$ of $\Gamma$, no node can be privileged infinitely many times without making any move in $\Gamma'$. It follows immediately from this definition that every finite execution is a fair-daemon-model execution.

1.1.2 Dolev’s read/write atomicity model

The above three computational models assume the composite atomicity. A single move (or atomic step) by a processor consists of reading registers of all its neighbors, making internal computations and then rewriting its own register (or registers). In 1993, Dolev et al., introduced a new type of computational model (we shall call it Dolev’s read/write atomicity model, or simply, the Dolev model) in their famous paper (Dolev et al., 1993) (cf. also (Collin & Dolev, 1994; Dolev, 2000; Dolev et al., 1990)). Their model reflects more truthfully a real distributed system. Firstly, it assumes the more real read/write atomicity. Under such an assumption, each atomic step in the system consists of internal computations and either a single read operation or a single write operation. Secondly, under this model, it is assumed that each processor in the system runs its own program indefinitely and at its own pace; and the running of the program has to follow the order of the statements in the program. Therefore, algorithms operating under the Dolev model have different looks from algorithms operating under composite atomicity models (e.g., one can compare Algorithm DM with Algorithm ADV in Section 2). Under the Dolev model, the behavior of the system under the action of the algorithm can still be described by a sequence of configurations $\Gamma = (\gamma_1, \gamma_2, \ldots)$. As in Dijkstra’s central daemon model, in any configuration $\gamma_i$ ($i \geq 1$), a unique processor of the system is selected by a daemon to make a move in the $i^{th}$ step $\gamma_i \rightarrow \gamma_{i+1}$, which changes the system configuration from $\gamma_i$ to $\gamma_{i+1}$. However, we should point out that due to the content of the algorithm and the way in which the algorithm is executed, the selection by this daemon is no longer random here under the Dolev model. In other words, any execution of the algorithm under the Dolev model has to obey certain restrictions. (In Section 2, a precise definition of an execution under the Dolev model, given for a certain class of algorithms in which we are interested, will almost completely reflect such restrictions.)

1.2 Self-stabilization

A distributed algorithm and a subset of executions (of the algorithm under a certain computational model), called legal executions, are designed to solve a specific problem such as the shortest path problem, the bridge-finding problem, or the mutual exclusion problem, etc. Legal executions are designed in such a way that if the system is engaged in any legal execution, the solution of the problem can be easily seen. For instance, for the single-source shortest path problem, the legal executions are so designed that in every configuration of any legal execution, a shortest path between any processor and the source can be identified. A configuration $\gamma$ is called a safe configuration or legitimate configuration (with respect to an algorithm under a particular computational model) if any execution (of the algorithm under that computational model) starting with $\gamma$ is a legal execution.

An algorithm is self-stabilizing under a certain computational model (with respect to a particular problem) if any execution of the algorithm under that computational model contains a safe configuration. A self-stabilizing algorithm under a certain computational model is silent if any execution of the algorithm under that computational model contains a configuration $\gamma$ such that in the suffix of
the execution starting with $\gamma$, the values stored in all the local variables and all the shared registers of all the processors never change. Most of the existing self-stabilizing algorithms (e.g., (Collin & Dolev, 1994; Ghosh et al., 1996a; Ghosh et al., 1996b; Ghosh et al., 1996c; Huang, 2005; Huang & Chen, 1992; Hsu & Huang, 1992; Huang & Lin, 2002; Huang et al., 2000; Huang et al., 2007; Huang et al., 2004; Ikeda et al., 2002; Shukla et al., 1995; Tsin, 2007; Turau, 2007; Turau & Hauck, 2009; Tzeng et al., 2007)) are silent.

1.3 Our contributions

In (Dolev et al., 1990), the following was mentioned: “It should be noted that one step of the distributed daemon in which $m$ processors move simultaneously can be simulated by our model: First let each of the $m$ processors read all its neighbors’ states, and then let them all move to their new states. Using this simulation, it is not hard to show that every mutual exclusion protocol that is self-stabilizing for read/write daemon is also self-stabilizing for distributed daemon”. The above comment concerns the mutual exclusion problem in particular, and hence the following question naturally arises: Can the point of view and the logic of the above “simulation” approach be applied to any problem for which there exists a self-stabilizing algorithm under the Dolev model? In other words, for any such problem, can the above “simulation” approach be used to show that a self-stabilizing algorithm under the Dolev model, when adapted to the distributed daemon model, is also self-stabilizing?

Since we have found a counterexample (see Appendix) to show that a self-stabilizing algorithm under the Dolev model may not be self-stabilizing when adapted to the central daemon model and hence may not be self-stabilizing when adapted to the distributed daemon model, the answer to the above question is, to one’s surprise, negative. This result shown by the counterexample also implies that applying the above simulation to an execution under the central daemon model may not result in an execution under the Dolev model. As a matter of fact, we are able to show that applying the above simulation to even an execution under the fair daemon model may not result in an execution under the Dolev model. Consequently, the above “simulation” approach cannot be used to show that a self-stabilizing algorithm under the Dolev model, when adapted to the fair daemon model, is also self-stabilizing.

By the above discussion, we have clarified that it is not a matter of course whether or not a self-stabilizing algorithm under the Dolev model, when adapted to a composite atomicity model, is also self-stabilizing. In this paper, we will show that if a self-stabilizing algorithm under the Dolev model has one of two particular forms, then it is also self-stabilizing when adapted to one of the composite atomicity models, the fair daemon model. To show this result, we need to develop a new simulation which is different from the above simulation suggested in (Dolev et al., 1990). Since most existing self-stabilizing algorithms under the Dolev model (e.g., (Collin & Dolev, 1994; Dolev et al., 1990; Dolev et al., 1993; Huang, 2005; Huang et al., 2004)) have the above-mentioned forms, our results imply that they are all self-stabilizing when adapted to the fair daemon model.

1.4 Organization of this paper

The rest of this paper is organized as follows: In Section 2, an abstract form (Algorithm DM) of a certain class of silent self-stabilizing algorithms under the Dolev model is presented, and its adapted version (Algorithm ADV) to composite atomicity models is given. In Section 3, an example is exhibited to illustrate the adaptation in Section 2. Then in Section 4 it is verified that the adapted algorithm (Algorithm ADV) is self-stabilizing under the fair daemon model. In Section 5, we consider another abstract form (Algorithm DMM) of a certain class of silent self-stabilizing algorithms under the Dolev model, and its adapted version (Algorithm ADV’) to composite atomicity models. Then we sketch
a proof to show that Algorithm ADV′ is self-stabilizing under the fair daemon model. Finally, we provide an example to illustrate the discussion in this section.

2 Adaptation from Algorithm DM into Algorithm ADV

As before, let $G = (V, E)$ denote a distributed system and for each $x \in V$, let $N(x)$ denote the set of all neighbors of $x$. Suppose Algorithm DM is a silent self-stabilizing algorithm under the Dolev model, with which $G$ is equipped to solve a certain problem. Suppose Algorithm DM has the following abstract form:

Algorithm DM
{For any node $x$}
1 repeat forever
2 for each $y \in N(x)$ do
3 read ($r_{yx} := w_y$)
4 endfor
5 if $w_x \neq A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x})$ then
6 write ($w_x := A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x})$)
7 endif
8 endrepeat

where

1) $w_x$ is a shared register of $x$, in which $x$ writes and from which all neighbors of $x$ read,
2) $r_{yx}$ is a local variable of $x$, in which $x$ stores the value that it reads from the shared register $w_y$ of its neighbor $y$,
3) $\delta(x)$ is the number of neighbors of $x$, and
4) $y_1, y_2, \ldots, y_{\delta(x)}$ are all the neighbors of $x$.

Since Algorithm DM is a silent self-stabilizing algorithm under the Dolev model, a legal execution of Algorithm DM is defined to be an execution in any configuration of which the guard condition $[w_x \neq A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x})]$ for each processor $x$ is always false, and $r_{yx} = w_y$ always holds for each processor $x$ and each neighbor $y$ of $x$. Hence a safe configuration with respect to Algorithm DM is a configuration in which both conditions $[\forall x \in V, w_x = A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x})]$ and $[\forall x \in V$ and $\forall y \in N(x), r_{yx} = w_y]$ hold.

As mentioned previously in Section 1, due to the content of the algorithm and the way in which the algorithm is executed, any execution of Algorithm DM under the Dolev model has to obey certain restrictions. For the rigor of our later proofs, we give here a precise definition to an execution of Algorithm DM under the Dolev model: A sequence $\Gamma = (\gamma_1, \gamma_2, \ldots)$ of configurations is called an execution of Algorithm DM under the Dolev model if:

1) for each $i \geq 1$, $\gamma_{i+1}$ is obtained from $\gamma_i$ after a processor makes either a single read operation or a single write operation (the transition $\gamma_i \rightarrow \gamma_{i+1}$ from $\gamma_i$ to $\gamma_{i+1}$ is called the $i^{\text{th}}$ step of $\Gamma$),
2) in $\Gamma$, each processor makes a read operation infinitely often,
3) in $\Gamma$, once a processor makes a read operation, it must complete a full round of reading all its neighbors (i.e., it must complete a full execution of the loop from statement 2 to statement 4 in Algorithm DM), and
(4) in \( \Gamma \), if the guard condition \( [w_x \neq A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_\delta(x)x})] \) is evaluated to be true right after any processor \( x \) completes a full round of reading all its neighbors, then the write operation “write \( (w_x := A_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_\delta(x)x})) \)” has to follow as the next move by \( x \).

Next, we introduce Algorithm ADV, which is a naturally-adapted version of Algorithm DM to composite atomicity models, and operates also in \( G \):

**Algorithm ADV**

{For any node \( x \)}

\[ R1 : w'_x \neq A_x(w'_{y_1}, w'_{y_2}, \ldots, w'_{y_\delta(x)}) \rightarrow w'_x := A_x(w'_{y_1}, w'_{y_2}, \ldots, w'_{y_\delta(x)}) \]

where

1. \( w'_x \) is the only shared register of \( x \),
2. \( \delta(x) \) is the number of neighbors of \( x \), and
3. \( y_1, y_2, \ldots, y_\delta(x) \) are all the neighbors of \( x \).

Note that in order for Algorithm ADV to solve the same problem as Algorithm DM does, a legal execution of Algorithm ADV should be defined to be an execution in any configuration of which no node can be privileged. As a consequence, the legal executions of Algorithm ADV are precisely all the one-configuration executions, and the safe configurations are precisely all the configurations in which no node can be privileged. Thus, under any of the three composite atomicity models introduced in the introduction, Algorithm ADV is self-stabilizing if and only if all its executions are finite.

### 3 An illustration

Suppose \( G = (V, E) \) is a distributed system in which there is a unique special node \( r \), called root. The following Algorithm DFS_DM is essentially the same as the DFS algorithm in (Collin & Dolev, 1994). Using the result in (Collin & Dolev, 1994), one can easily convince oneself that Algorithm DFS_DM is a silent self-stabilizing algorithm under the Dolev model, with which \( G \) can be equipped to solve the Depth First Search (DFS) spanning tree problem.

**Algorithm DFS_DM**

{For the root \( r \)}

1 repeat forever
2 for each \( y \in N(r) \) do
3 \( \text{read (} \text{read\_path}_y := \text{write\_path}_y \text{)} \)
4 endfor
5 if \( \text{write\_path}_r \neq \bot \) then
6 \( \text{write (} \text{write\_path}_r := \bot \text{)} \)
7 endif
8 endrepeat

{For any node \( x \neq r \)}

1 repeat forever
2 for each \( y \in N(x) \) do
3 \( \text{read (} \text{read\_path}_{yx} := \text{write\_path}_y \text{)} \)
In the above algorithm, for root or non-root $x$,

1) $\text{write}_{x}$ is a shared register of $x$, in which $x$ writes and from which all neighbors of $x$ read,

2) $\text{read}_{y}$ is a local variable of $x$, in which $x$ stores the value that it reads from the shared register $\text{write}_{y}$ of its neighbor $y$,

3) $\alpha_x$ is a pre-given ordering of all the edges incident to $x$, and $\alpha(x)$ is the rank of edge $\{x, y\}$ in the ordering $\alpha_x$,

4) the value of either $\text{write}_{x}$ or $\text{read}_{y}$ is a sequence of at most $N$ items, where $N \geq n$ (the number of nodes in $G$) is a constant positive integer, and each item in the sequence is either a positive integer or a special symbol $\bot$,

5) $<_{\text{lex}}$ is a lexicographical order over the set of all finite sequences of at most $N$ items, and $\bot$ is the minimal character (for example, $(\bot, 1) <_{\text{lex}} (\bot, 1, 1) <_{\text{lex}} (\bot, 2) <_{\text{lex}} (1)$),

6) $\text{read}_{y} \circ \alpha(y)$ is the concatenation of $\text{read}_{y}$ with $\alpha(y)$ (for example, if $\text{read}_{y} = (1, 2, 1) \text{ and } \alpha(y) = 3$, then $\text{read}_{y} \circ \alpha(y) = (1, 2, 1, 3)$), and

7) the notation $\text{right}_{N}(w)$ refers to the sequence of the $N$ least significant items of $w$ (for example, $\text{right}_{6}(\bot, 3, 2, 1, 2, 1, 1) = (3, 2, 1, 2, 1, 1)$ and $\text{right}_{6}(2, 1, 2, 1, 1) = (2, 1, 2, 1, 1)$).

Since Algorithm $\text{DFS}_{\text{DM}}$ is a silent self-stabilizing algorithm under the Dolev model, a legal execution of Algorithm $\text{DFS}_{\text{DM}}$ is defined to be an execution in any configuration of which the guard conditions “$\text{write}_{r} \neq \bot$” for the root $r$ and “$\text{write}_{x} \neq \min_{\text{lex}} \{\text{right}_{N}(\text{read}_{y} \circ \alpha(y)) \mid y \in N(x)\}$” for each processor $x \neq r$ are always false, and “$\text{read}_{y} = \text{write}_{y}$” always holds for each processor $x$ and each neighbor $y$ of $x$. Hence a legal execution of Algorithm $\text{DFS}_{\text{DM}}$ is an execution in any configuration of which both conditions $[\text{write}_{r} = \bot \text{ and } \forall y \in N(r), \text{read}_{y} = \text{write}_{y}]$ and $[\forall x \in V - \{r\}, \text{write}_{x} = \min_{\text{lex}} \{\text{right}_{N}(\text{read}_{y} \circ \alpha(y)) \mid y \in N(x)\}]$ and $\forall x \in V - \{r\}$ and $\forall y \in N(x)$, $\text{read}_{y} = \text{write}_{y}$ hold.

According to the adaptation in Section 2, the naturally-adapted version of Algorithm $\text{DFS}_{\text{DM}}$ to composite atomicity models is as follows:

**Algorithm $\text{DFS}_{\text{ADV}}$**

\{For the root $r$\}

$R0: \text{path}_{r} \neq \bot \rightarrow \text{path}_{r} := \bot$

\{For any node $x \neq r$\}

$R1: \text{path}_{x} \neq \min_{\text{lex}} \{\text{right}_{N}(\text{path}_{y} \circ \alpha(y)) \mid y \in N(x)\}$

$\rightarrow \text{path}_{x} := \min_{\text{lex}} \{\text{right}_{N}(\text{path}_{y} \circ \alpha(y)) \mid y \in N(x)\}$

Note that in the above algorithm, for root or non-root $x$,

1) $\text{path}_{x}$ is the only register of $x,$
2) the value of path\(_x\) is a sequence of at most \(N\) items, where \(N \geq n\) (the number of nodes in \(G\)) is a constant positive integer, and each item in the sequence is either a positive integer or a special symbol \(\perp\), and

3) the notations \(\alpha_{x}\), \(\alpha_{x}(y)\), \(\prec_{lex}\), \(\right_{N}(\cdot)\) and \(\text{path}_{y} \circ \alpha_{y}(x)\) have the same meanings as in Algorithm DFS\(_{DM}\).

In order for Algorithm DFS\(_{ADV}\) to solve the DFS spanning tree problem as Algorithm DFS\(_{DM}\) does, a legal execution of Algorithm DFS\(_{ADV}\) should be defined to be an execution in any configuration of which no node can be privileged. As a consequence, the legal executions of Algorithm DFS\(_{ADV}\) are precisely all the one-configuration executions, and the safe configurations with respect to Algorithm DFS\(_{ADV}\) are precisely all the configurations in which no node can be privileged, i.e., the condition \([\text{path}_{r} = \perp \text{ and } \forall x \in V - \{r\}, \text{path}_{x} = \min_{\prec_{lex}} \{\right_{N}(\text{path}_{y} \circ \alpha_{y}(x)) \mid y \in N(x)\}]\) holds. As mentioned previously, under any of the above three composite atomicity models introduced in the introduction, Algorithm DFS\(_{ADV}\) is self-stabilizing if and only if all its executions are finite.

4 Correctness of Algorithm ADV under the fair daemon model

As mentioned previously, one of our main contributions in this paper is to show that Algorithm ADV is self-stabilizing under the fair daemon model. In this section, we will prove this by contradiction. So we first suppose Algorithm ADV is not self-stabilizing under the fair daemon model. Hence there exists an infinite execution \(\Gamma' = (\gamma'_1, \gamma'_2, \ldots)\) of Algorithm ADV under the fair daemon model. By using the process below, we can simulate \(\Gamma'\) by an execution \(\Gamma\) of Algorithm DM under the Dolev model:

1. For each \(i = 1, 2, \ldots\), let \(NP_i\) be the set of the nodes which are not privileged in \(\gamma'_i\), and \(PM_i\) be the set of the nodes which are privileged in \(\gamma'_i\) and chosen by the fair daemon to make moves in the step \(\gamma'_i \rightarrow \gamma'_{i+1}\).

2. Define \(\gamma_1\) to be the configuration of the system \(G\) equipped with Algorithm DM such that for each \(x \in V\), the value of \(w_x\) in \(\gamma_1\) is equal to the value of \(w'_x\) in \(\gamma'_1\); and for each \(x \in V\) and each neighbor \(y\) of \(x\), the value of \(r_{yx}\) in \(\gamma_1\) is equal to the value of \(w'_y\) in \(\gamma'_1\).

3. Starting with the configuration \(\gamma_1\),

   (a) let nodes in \(NP_1 \cup PM_1\) one by one make read operations (i.e., let them execute statements 2-4 of Algorithm DM), and then

   (b) let nodes in \(PM_1\) one by one make write operations (i.e., let them execute statements 5-7 of Algorithm DM).

(Note that the behavior of the system \(G\) under all these operations in (a) and (b) above can be represented by a finite sequence \(S_1\) of configurations starting with \(\gamma_1\). Let the last configuration of \(S_1\) be \(\gamma_2\), that is, \(S_1 = (\gamma_1, \ldots, \gamma_2)\). One can easily see that for each node \(x \in V\), the value of \(w_x\) in \(\gamma_2\) is equal to the value of \(w'_x\) in \(\gamma'_2\).)

4. Similarly, for any \(i \geq 2\), starting with the configuration \(\gamma_i\),

   (a) let nodes in \(NP_i \cup PM_i\) one by one make read operations (i.e., let them execute statements 2-4 of Algorithm DM), and then
(b) let nodes in \( PM_i \) one by one make write operations (i.e., let them execute statements 5-7 of Algorithm DM).

(Note that the behavior of the system \( G \) under all these operations in (a) and (b) above can be represented by a finite sequence \( S_t \) of configurations starting with \( \gamma_i \). Let the last configuration of \( S_t \) be \( \gamma_{i+1} \), that is, \( S_t = (\gamma_i, \ldots, \gamma_{i+1}) \). One can easily see that for each node \( x \in V \), the value of \( w_x \) in \( \gamma_{i+1} \) is equal to the value of \( w'_x \) in \( \gamma_{i+1}' \).)

5. From the above, we finally obtain an infinite sequence \( \Gamma = (\gamma_1, \ldots, \gamma_2, \ldots, \gamma_3, \ldots) \) of configurations of \( G \) equipped with Algorithm DM.

Claim. \( \Gamma \) is an execution of Algorithm DM under the Dolev model.

Proof of claim.

(1) According to the above construction process of \( \Gamma \), it is obvious that except for the initial configuration \( \gamma_1 \), any configuration in \( \Gamma \) is obtained from the preceding configuration after a processor makes either a single read operation or a single write operation.

(2) Suppose there exists a processor \( x \) that makes a read operation only a finite number of times in \( \Gamma \). Then there exists a configuration \( \gamma_t \) in \( \Gamma \) such that \( x \) can not make any read operation after \( \gamma_t \). According to the construction process of \( \Gamma \), for all \( l \geq t \), \( x \notin NP_l \cup PM_l \). Thus, \( x \) is privileged, but is not selected by the fair daemon to make any move in the suffix \( (\gamma'_t, \gamma'_{t+1}, \ldots) \) of \( \Gamma' \). This contradicts the fairness of \( \Gamma' \). Therefore, any processor makes a read operation infinitely often in \( \Gamma \).

(3) According to the construction process of \( \Gamma \), it is obvious that in \( \Gamma \) once a processor makes a read operation, it will complete a full round of reading all its neighbors.

(4) According to the construction process of \( \Gamma \), if in \( \Gamma \) any processor \( x \) completes a full round of reading all its neighbors, then it must do so in \( S_i \), for some \( i = 1, 2, \ldots \). Thus, \( x \in NP_{i} \cup PM_{i} \). If \( x \in NP_{i} \), then the guard condition \([w_x \neq A_x(r_{y_1,x}, r_{y_2,x}, \ldots, r_{y_{(i+1)x},x})]\) in statement 5 of Algorithm DM cannot be true right after \( x \) completes a full round of reading all its neighbors. If \( x \in PM_{i} \), then the guard condition \([w_x \neq A_x(r_{y_1,x}, r_{y_2,x}, \ldots, r_{y_{(i+1)x},x})]\) is evaluated to be true right after \( x \) completes a full round of reading all its neighbors. According to the construction process of \( \Gamma \), the write operation “write \((w'_x := A_x(r_{y_1,x}, r_{y_2,x}, \ldots, r_{y_{(i+1)x},x}, w_x))\)” will follow as the next move by \( x \).

From all the above, we can conclude that \( \Gamma \) is an execution of Algorithm DM under the Dolev model.

Since \( \Gamma \) is an execution of Algorithm DM and Algorithm DM is a silent self-stabilizing algorithm under the Dolev model, there exists a safe configuration \( \gamma \) in \( \Gamma \) such that in the suffix \( \Gamma \) of \( \Gamma \) that starts with \( \gamma \), the values stored in all the shared registers and local variables of all the nodes never change. Hence in \( \Gamma \) no node can make any write operation. In view of the construction process of \( \Gamma \), \( \gamma \in S_k \) for some \( k = 1, 2, \ldots \). Hence in \( S_{k+1} \) no node can make any write operation. However, since \( \Gamma' = (\gamma'_1, \gamma'_2, \ldots) \) is an infinite execution, there must be nodes selected by the fair (distributed) daemon to make moves in the step \( \gamma'_{k+1} \rightarrow \gamma'_{k+2} \). In view of the construction process of \( \Gamma \), these nodes must make write operations in \( S_{k+1} \). A contradiction occurs. Therefore, our original supposition that Algorithm ADV is not self-stabilizing under the fair daemon model is false, and hence we obtain the following theorem.

Theorem 1 Algorithm ADV is self-stabilizing under the fair daemon model.
5 Adaptation from Algorithm DMM into Algorithm ADV'

In the preceding section, we have shown that if Algorithm DM is a silent self-stabilizing algorithm under the Dolev model, then Algorithm ADV, a naturally adapted version of Algorithm DM, is a silent self-stabilizing algorithm under the fair daemon model. Note that in any silent self-stabilizing algorithm under the Dolev model that has the form of Algorithm DM, each processor only maintains a unique shared register, which can be read by all its neighbors. However, in the existing literature, there are also silent self-stabilizing algorithms under the Dolev model (e.g., (Dolev et al., 1993; Huang, 2005; Huang et al., 2004)) that have the form of Algorithm DMM exhibited below, in which each processor maintains an individual register for each of its neighbors.

Algorithm DMM
{For any node $x$}
1 repeat forever
2 for each $y \in N(x)$ do
3 read ($r_{yx} := w_{yx}$)
4 endfor
5 for each $y \in N(x)$ do
6 if $w_{xy} \neq B_x(r_{y1x}, r_{y2x}, \ldots, r_{y\delta(x)x})$ then
7 write ($w_{xy} := B_x(r_{y1x}, r_{y2x}, \ldots, r_{y\delta(x)x})$)
8 endif
9 endfor
10 endrepeat

where

1) $w_{xy}$ is a shared register of $x$, in which $x$ writes and from which its neighbor $y$ reads,

2) $r_{yx}$ is a local variable of $x$, in which $x$ stores the value that it reads from the shared register $w_{yx}$ of its neighbor $y$,

3) $\delta(x)$ is the number of neighbors of $x$, and

4) $y_1, y_2, \ldots, y_{\delta(x)}$ are all the neighbors of $x$.

Note that Algorithm DMM can be naturally adapted to composite atomicity models, and the resulting adapted version is as follows:

Algorithm ADV'
{For any node $x$}
$R1 : w'_x \neq B_x(w'_{y1}, w'_{y2}, \ldots, w'_{y\delta(x)}) \rightarrow w'_x := B_x(w'_{y1}, w'_{y2}, \ldots, w'_{y\delta(x)})$

where

1) $w'_x$ is the only shared register of $x$,

2) $\delta(x)$ is the number of neighbors of $x$, and

3) $y_1, y_2, \ldots, y_{\delta(x)}$ are all the neighbors of $x$. 
In view of the previous discussion in this paper, the following question comes out naturally: If Algorithm DMM is a silent self-stabilizing algorithm under the Dolev model, is Algorithm ADV a silent self-stabilizing algorithm under the fair daemon model?

The answer to the above question is affirmative, as we now sketch its proof: First, we modify Algorithm DMM (in which each processor maintains for all its neighbors only a unique shared register) in a natural way into Algorithm DM’ (in which each processor maintains for each of its neighbors an individual register).

**Algorithm DM’**

{For any node \( x \)}

1. repeat forever
2. for each \( y \in N(x) \) do
3. read \((r_{yx} := w_y)\)
4. endfor
5. if \( w_x \neq B_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x})\) then
6. write \((w_x := B_x(r_{y_1x}, r_{y_2x}, \ldots, r_{y_{\delta(x)}x}))\)
7. endif
8. endrepeat

where

1. \( w_x \) is a shared register of \( x \), in which \( x \) writes and from which all neighbors of \( x \) read,
2. \( r_{yx} \) is a local variable of \( x \), in which \( x \) stores the value that it reads from the shared register \( w_y \) of its neighbor \( y \),
3. \( \delta(x) \) is the number of neighbors of \( x \), and
4. \( y_1, y_2, \ldots, y_{\delta(x)} \) are all the neighbors of \( x \).

Next, by taking the two steps below, we can simulate any execution \( \Gamma \) of Algorithm DM’ under the Dolev model by an execution \( \Gamma^* \) of Algorithm DMM under the Dolev model:

1. For any step in the execution \( \Gamma \) of Algorithm DM’, if this step is a read operation performed by node \( x \) on its neighbor node \( y \) (i.e., it is an execution of statement 3 of Algorithm DM’ by \( x \)), then, corresponding to this step, we assign a step to the execution \( \Gamma^* \) of Algorithm DMM, in which node \( x \) performs exactly the same read operation on node \( y \) (i.e., \( x \) executes statement 3 of Algorithm DMM).
2. For any step in the execution \( \Gamma \) of Algorithm DM’, if this step is a write operation performed by node \( x \) (i.e., it is an execution of statements 5-7 of Algorithm DM’ by \( x \)), then, corresponding to this step, we assign to the execution \( \Gamma^* \) of Algorithm DMM those steps which are the obvious corresponding write operations (in an execution of statements 5-9 of Algorithm DMM) performed by node \( x \).

Of course, although not difficult, it is crucial to verify the execution \( \Gamma^* \) so defined to be indeed an execution of Algorithm DMM under the Dolev model. Then, via this simulation, one can easily see that under the Dolev model, if Algorithm DMM is a silent self-stabilizing algorithm, then Algorithm DM’ is also a silent self-stabilizing algorithm. Since Algorithm DM’ has the form of Algorithm DM in the preceding section, and Algorithm ADV’ is exactly the resulting algorithm after we apply the adaptation in Section 2 to Algorithm DM’, it follows from Theorem 1 that Algorithm ADV’ is a silent self-stabilizing algorithm under the fair daemon model.
As an illustration of the above discussion in this section, a slightly revised version of the shortest path algorithm in (Huang, 2005) can be used.

**Algorithm SP\_DMM**

\{For the source $s$\}

1. repeat forever
2. for each $y \in N(s)$ do
   3. read ($r_{ys} := d_{ys}$)
   4. endfor
3. for each $y \in N(s)$ do
   4. if $d_{sy} \neq 0$ then
   5. write ($d_{sy} := 0$)
   6. endif
   7. endfor
8. endrepeat

\{For node $x \neq s$\}

1. repeat forever
2. for each $y \in N(x)$ do
   3. read ($r_{yx} := d_{yx}$)
   4. endfor
5. for each $y \in N(x)$ do
   6. if $d_{xy} \neq \min_{z \in N(x)}(r_{zx} + w(x, z))$ then
   7. write ($d_{xy} := \min_{z \in N(x)}(r_{zx} + w(x, z))$)
   8. endif
   9. endfor
10. endrepeat

In the above algorithm, for any node $x$, and for any $y \in N(x)$,

1) $d_{xy}$ is a shared register of $x$, in which $x$ writes and from which $y$ reads,

2) $r_{yx}$ is a local variable of $x$, in which $x$ stores the value that it reads from the shared register $d_{yx}$ of its neighbor $y$, and

3) $w(x, y)$ is the weight (or length) pre-assigned to the edge $\{x, y\}$ connecting $x$ and $y$.

In view of the results in (Huang, 2005), one can be easily convinced that Algorithm SP\_DMM is a silent self-stabilizing algorithm under the Dolev model. Note that Algorithm SP\_DMM has the form of Algorithm DMM. So if we apply the above adaptation in this section to Algorithm SP\_DMM, the following adapted version to composite atomicity models can be obtained:

**Algorithm SP\_ADV’**

\{For the source $s$\}

1. $R0 : d_s \neq 0 \rightarrow d_s := 0$

\{For any node $x \neq s$\}

1. $R1 : d_x \neq \min_{z \in N(x)}(r_z + w(x, z)) \rightarrow d_x := \min_{z \in N(x)}(r_z + w(x, z))$

A slight modification of Algorithm SP\_DMM (in which each processor maintains for each of its neighbors an individual register) gives us Algorithm SP\_DMM’ (in which each processor maintains for all its
neighbors only a unique shared register).

**Algorithm SP DM’**
{For the source $s$}
1 repeat forever
2 for each $y \in N(s)$ do
3 read ($r_{ys} := d_y$)
4 endfor
5 if $d_s \neq 0$ then
6 write ($d_s := 0$)
7 endif
8 endrepeat

{For node $x \neq s$}
1 repeat forever
2 for each $y \in N(x)$ do
3 read ($r_{yx} := d_y$)
4 endfor
5 if $d_x \neq \min_{z \in N(x)} (r_{xz} + w(x, z))$ then
6 write ($d_x := \min_{z \in N(x)} (r_{xz} + w(x, z))$)
7 endif
8 endrepeat

In view of the previous discussion in this section, since Algorithm SP DM is a silent self-stabilizing algorithm under the Dolev model, Algorithm SP DM’ is also a silent self-stabilizing algorithm under the Dolev model. Note that Algorithm SP ADV’ is exactly the resulting algorithm after we apply the adaptation in Section 2 to Algorithm SP DM’, so it follows from Theorem 1 that Algorithm SP ADV’ is a silent self-stabilizing algorithm under the fair daemon model.

**Appendix**

In this appendix, we give a counterexample to show that a self-stabilizing algorithm under the Dolev model, when adapted to the central daemon model, may no longer be self-stabilizing. The counterexample which we are going to give is the Algorithm DFS DM and its adapted version Algorithm DFS ADV presented previously in Section 3.

Consider now a distributed system $G$ consisting of 6 nodes as illustrated in Figure 1. The number $N$ in Algorithm DFS ADV, with which $G$ is equipped, is now set to 6. In Table 1 is exhibited a beginning portion of an infinite cyclic execution $\Gamma = (\gamma_1, \gamma_2, \ldots)$ of Algorithm DFS ADV under the central daemon model, which has a period of 13 (note that $\gamma_1 = \gamma_{13}$). The shaded grid in each configuration of $\Gamma$ in the table indicates the privileged node selected by the central daemon to make a move. As mentioned in the end of Section 3, Algorithm DFS ADV is self-stabilizing under the central daemon model if and only if all its executions under the central daemon model are finite. Now that an infinite execution of Algorithm DFS ADV under the central daemon model is found to exist, it is shown that Algorithm DFS ADV is not a self-stabilizing algorithm under the central daemon model.
Figure 1: A distributed system $G$ labelled with edge ranks.

References


Table 1: A beginning portion of an infinite cyclic execution $\Gamma$ of Algorithm DFS$_{ADV}$ under the central daemon model for the system in Figure 1.

<table>
<thead>
<tr>
<th>$r_i$</th>
<th>path $a$</th>
<th>path $c$</th>
<th>path $a$</th>
<th>path $c$</th>
<th>path $d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_1$</td>
<td>(1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>$r_2$</td>
<td>(1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>$r_3$</td>
<td>(1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>$r_4$</td>
<td>(1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>$r_5$</td>
<td>(1,1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1,1)</td>
<td>(1,1,1,1,1,1,1,1,1,1)</td>
<td></td>
</tr>
</tbody>
</table>


