Correctness of Self-Stabilizing Algorithms Under the Dolev Model When Adapted to Composite Atomicity Models

Chih-Yuan Chen, Department of Computer Science and Information Engineering, Tao-yuan Innovation Institute of Technology, Chung-Li, Taiwan
Cheng-Pin Wang, General Education Center, Tzu Chi College of Technology, Hualien, Taiwan
Tetz C. Huang, Department of Computer Science and Engineering, Yuan-Ze University, Chung-Li, Taiwan
Ji-Cherng Lin, Department of Computer Science and Engineering, Yuan-Ze University, Chung-Li, Taiwan

ABSTRACT
In this paper, the authors 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 the authors 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, the authors’ results imply that they are all self-stabilizing when adapted to the fair daemon model.

Keywords: Adaptation of Algorithm, Composite Atomicity, Fair Daemon Model, Read/Write Atomicity, Silent Self-Stabilizing 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

DOI: 10.4018/ijalr.2012100102
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 uninterrupted. 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:

condition part → 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
Related Content

Forward and Backward Chaining with P Systems
Sergiu Ivanov, Artiom Alhazov, Vladimir Rogojin and Miguel A. Gutiérrez-Naranjo (2014). *Natural Computing for Simulation and Knowledge Discovery* (pp. 149-158).
[www.igi-global.com/chapter/forward-and-backward-chaining-with-p-systems/80062?camid=4v1a](www.igi-global.com/chapter/forward-and-backward-chaining-with-p-systems/80062?camid=4v1a)

Algorithms and Methods Inspired from Nature for Solving Supply Chain and Logistics Optimization Problems: A Survey
[www.igi-global.com/article/algorithms-and-methods-inspired-from-nature-for-solving-supply-chain-and-logistics-optimization-problems/118156?camid=4v1a](www.igi-global.com/article/algorithms-and-methods-inspired-from-nature-for-solving-supply-chain-and-logistics-optimization-problems/118156?camid=4v1a)

A Flexible Bio-Signal Based HMI for Hands-Free Control of an Electric Powered Wheelchair
[www.igi-global.com/article/a-flexible-bio-signal-based-hmi-for-hands-free-control-of-an-electric-powered-wheelchair/103856?camid=4v1a](www.igi-global.com/article/a-flexible-bio-signal-based-hmi-for-hands-free-control-of-an-electric-powered-wheelchair/103856?camid=4v1a)
Inferring Gene Regulatory Networks from Genetical Genomics Data
[www.igi-global.com/chapter/inferring-gene-regulatory-networks-genetical/38232?camid=4v1a](www.igi-global.com/chapter/inferring-gene-regulatory-networks-genetical/38232?camid=4v1a)