Fault-tolerant Implementations of the Atomic-state Communication Model in Weaker Networks *

Colette Johnen[1] and Lisa Higham[2]

[1] LRI, Univ. Paris-Sud, CNRS, F-91405 Orsay, France colette@lri.fr
[2] Computer Science Department, University of Calgary, Canada higham@ucalgary.ca

There is a proliferation of models for distributed computing, consisting of both shared memory and message passing paradigms. Different communities adopt different variants as the "standard" model for their research setting. Since subtle changes in the communication model can result in significant changes to the solvability/unsolvability or to the complexity of various problems, it becomes imperative to understand the relationships between the many models. The situation becomes even more complicated when additional requirements such as fault-tolerance are added to the mix. This motivates us to determine exactly under what circumstances a program designed for one model and delivering some set of additional guarantees can be converted into an "equivalent" programs for a different model while delivering comparable guarantees. Once these relationships are understood, they can be exploited in system design.

Our work addresses this question for networks of processors that communication by locally shared registers. A network that uses locally shared registers can be modelled by a graph where nodes represent processors and there is an edge between two nodes if and only if the corresponding processors communicate directly by reading or writing registers shared between them. Two variants are defined by specifying whether the registers are single-writer/multi-reader and located at the nodes (called state models) or single-writer/ single-reader and located on the edges (called link models).

The shared registers used by the communicating processors further distinguishes possible models. Lamport [4] defined three models of single-writer/multi-reader registers, differentiated by the possible outcome of read operations that overlap concurrent write operations. These three register types, in order of increasing power, are called safe, regular, and atomic. Program design is easier assuming atomic registers rather than weaker registers but the hardware implementation of an atomic register is costlier than the implementation of one of the weaker ones.

By specifying either state or link communication, via shared registers that are either regular, atomic, or safe we arrive at six different network models that use locally shared registers. For example, the atomic-state model has atomic registers located at the nodes of the network. The other models are named similarly. An algorithm for any one of these networks could provide some fault tolerance. So, we consider a third parameter, namely, wait-freedom, which captures tolerance of stopping failures of components of the network, or self-stabilization, which captures recovery of the network from transient errors of its components.

We seek to determine under what conditions it is possible to transform a wait-free (respectively, self-stabilizing) solution to a given problem under one of these models into a wait-free solution (respectively, self-stabilizing) solution under another of the six models.

In an earlier paper [2], we proved that a wait-free compiler from atomic-state systems to atomic-link systems requires that if two processors, a and b, each share a register with a third processor c, then a and b must themselves share a register. But, in a network, processors that are not neighbours cannot share a register. A consequence of this essential distinction between networks and globally shared memory systems is the impossibility: "There is no wait-free compiler from atomic-state systems to regular-state systems for the same network graph for any network graph that is not complete". We also presented a self-stabilizing compiler from network graphs where neighbours communicate via atomic-state registers to systems where neighbours communicate via atomic-link registers [2]. This compiler, however, had some shortcomings. It is not a silence-preserving compiler; it requires that each processor in the atomic-state system being implemented executes an atomic-state read operation infinitely often; the implementation of the atomic-state write operation is not wait-free; the implementation of the atomic-state read operation is not even obstruction-free. Furthermore,

* This work was partially supported by the program ACI "security and dependability" FRAGILE and by NSERC (canada) Discovery Grant.

the proof of correctness failed to characterize the legitimate configurations: instead it only established that all computations of the compiled algorithm are *eventually* linearizable.

Contributions Our principal result is a self-stabilizing compiler from the atomic-state model to the regular-state model. This compiler is also *silent*. That is, if, once registers are stabilized, the atomic-state algorithm does not require the participation of neighbours, then the transformed regular-state algorithm also does not require the participation of neighbours. As a consequence, our compiler does not add significant overhead to communication. The code and the self-stabilization proof is presented in a technical report [3]. Our compiler has some additional appealing properties: The size of each shared regular register used by the compiled algorithm is $log(M) + 1 + log(B)$ bits where M is the number of processor states of the initial algorithm and B is greater than the network degree. The compiled algorithm has strong progress guarantees. Specifically, the implementation of any write operation is wait-free. The implementation of a read is not wait-free, but it is obstruction-free.

For all the remaining relationships (both possibilities and impossibilities) among the four models that use atomic and regular registers under either self-stabilizing and wait-free requirements, we either observe that they have been answered by existing research or we show how they can be derived from combinations of earlier results. Thus, our compiler closes the proposed questions among four of the six models. These results are summarized in the following figure.

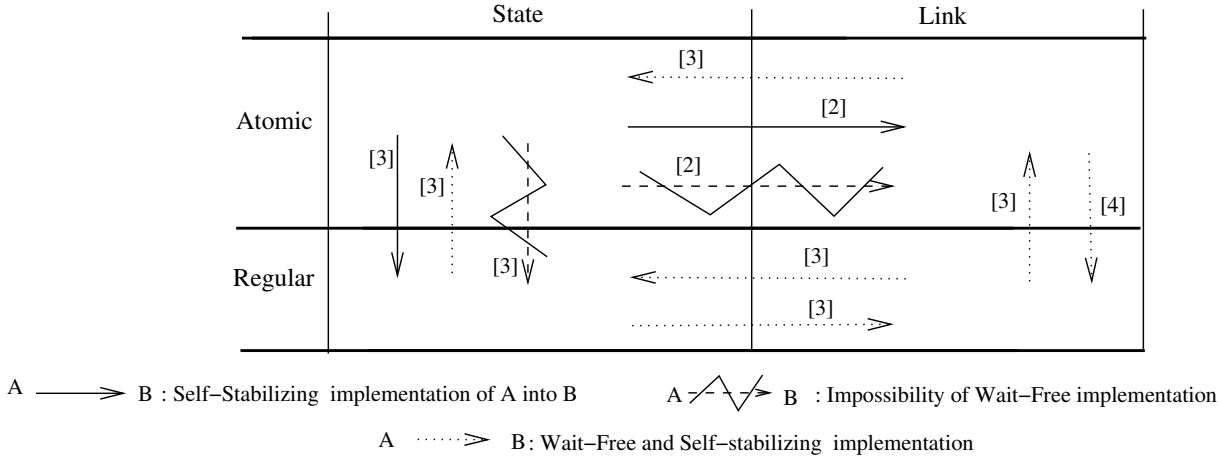

Fig. 1. Transformations between network models

What remains open is whether or not there is a self-stabilizing compiler from networks (state or link) with regular registers, to the corresponding network with only safe registers. Interestingly, Lamport's construction of single-writer single-reader single-bit regular registers from single-writer single-reader single-bit safe registers [4] fails to be self-stabilizing [1]. We conjecture that this shortcoming can be rectified at the expense of wait-freedom.

References

1. JH Hoepman, M Papatriantafilou, and P Tsigas. Self-stabilization of wait-free shared memory objects. *Journal of Parallel and Distributed Computing*, 62(5):818–842, 2002.
2. L Higham and C Johnen. Relationships between communication models in networks using atomic registers. In *IPDPS'2006, the 20th IEEE International Parallel & Distributed Processing Symposium*, 2006.
3. L. Higham and C. Johnen. Self-stabilizing implementation of atomic register by regular register in networks framework. Technical Report 1449, L.R.I, 2006.
4. L Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–101, 1986.