Fault-Tolerant Implementations of Atomic Registers by Safe Registers in Networks

Colette Johnen
LRI, Univ. Paris-Sud, CNRS
colette@lri.fr

Lisa Higham
Computer Science Dept, U. Calgary, Canada
higham@ucalgary.ca

Categories G.4[Reliability and Robustness]; D.1.3[Distributed Programming]
General Terms Algorithms, Theory, Design

A distributed network can be modelled by a communication graph $G = (V, E)$ where V is a set of processors and an edge $\langle pq \rangle \in E$ if and only if processors p and q can communicate directly. Several variants have been defined depending on the precise meaning of "communicate directly". In the *state* network models, each processor owns a single-writer/multi-reader shared register, writable by the owner and readable by each neighbour. In the *link* network models, for each pair of processors joined by an edge, there are two single-writer/single-reader registers, each writable by one and readable by the other. Lamport [5] defined three kinds of registers depending on the semantics when READ and WRITE operations overlap: safe, regular and atomic. Thus, six different network models arise by specifying two parameter for the shared registers: strength in {atomic, regular, safe} and location in {link, state}. Our goal has been to determine possibilities and impossibilities for transforming an algorithmic solution designed for one variant into a solution for a weaker variant while retaining one or both of the fault tolerance properties wait-freedom and self-stabilization. In all six models the registers are single-writer, and so this descriptor is omitted in the following summary.
We previously reported the following algorithms and impossibilities for transforming between some of these models:
1. From atomic-state to atomic-link– a self-stabilizing compiler, and a proof that a wait-free one is impossible [1].
2. From atomic-state to regular-state– a self-stabilizing compiler that wait-free implements each WRITE operation and obstruction-free implements each READ operation, and a proof that it is impossible for both to be wait-free [2].
3. From regular-state to regular-link– there is a straightforward compiler that is wait-free and self-stabilizing [2,3].
4. From atomic-link to regular-link– there is a straightforward interpretation of Lamport's work [5] that constitutes a compiler that is wait-free and self-stabilizing [2,3].

PODC'08, August 18–21, 2008, Toronto, Ontario, Canada.

Our new contribution (see [4]) is a self-stabilizing and wait-free compiler from regular-link to safe-link networks. The relationship between these two networks is the same as that between single-reader regular registers and single-reader safe registers. We construct a single-reader regular register using only single-reader safe registers, in two steps, relying on an intermediate register type, called 1-regular, which behaves like a regular register provided any READ has at most one overlapping WRITE, but like a safe register otherwise.
Step 1: A WRITE of a single-reader 1-regular register is implemented by writing to each of three safe registers; a READ is implement by reading the three safe registers in the opposite order. Provided at most one WRITE overlaps any READ, at most one of the safe registers could be concurrently read and written. So two are "uncorrupted", which is enough to determine a correct final value for the READ to return.
Step 2: If at most one WRITE could overlap a READ, a 1-regular register would suffice in place of a regular register. This suggests that we try to avoid overlap by having more than one (three suffices) 1-regular register available for a writer and arranging communication from the reader to direct the writer which one to use. The writer indicates which register it actually used, by setting two of three possible pointers to point to it. The reader executes a loop three times, each time examining a pointer to choose one of the three registers, and returning the value of the chosen register. The algorithm ensures that at most one of the chosen values can be stale, and a trick ensures that a stale value is never selected as the final value returned by the READ.
By composing various compilers appropriately, we can implement algorithms designed for the atomic-state or the atomic-link (also called the Read-Write atomicity model) model on the safe-link model, while preserving self-stabilization.

References.

[1] L Higham and C Johnen. Relationships between communication models in networks using atomic registers. In *IPDPS'2006*, (Report 1419, LRI), 2006.
[2] L Higham and C Johnen. Self-stabilizing implementation of atomic register by regular register in networks framework. Report 1449, LRI, 2006.
[3] L Higham and C Johnen. Fault-Tolerant Implementations of the Atomic-State Communication Model in Weaker Networks. DISC 2007 BA: 485-487.
[4] C Johnen and L Higham. Self-stabilizing implementation of regular register by safe registers in link model. Report 1486, LRI, 2008. http://www.lri.fr.
[5] L Lamport. On interprocess communication. *Distributed Computing*, 1(2):77–101, 1986.