Registers

1986; Lamport, Vitanyi, Awerbuch

PAUL VITÁNYI
CWI, Amsterdam, Netherlands

Synonyms

Shared-memory (wait-free); Wait-free registers; Wait-free shared variables; Asynchronous communication hardware

Problem Definition

Consider a system of asynchronous processes that communicate among themselves by only executing read and write operations on a set of shared variables (also known as shared *registers*). The system has no global clock or other synchronization primitives. Every shared variable is associated with a process (called *owner*) which writes it and the other processes may read it. An execution of a write (read) operation on a shared variable will be referred to as a *Write* (*Read*) on that variable. A Write on a shared variable puts a value from a pre-determined finite domain into the variable, and a Read reports a value from the domain. A process that writes (reads) a variable is called a *writer* (*reader*) of the variable.

The goal is to construct shared variables in which the following two properties hold. (1) Operation executions are not necessarily atomic, that is, they are not indivisible but rather consist of atomic sub-operations, and (2) every operation finishes its execution within a bounded number of its own steps, irrespective of the presence of other operation executions and their relative speeds. That is, operation executions are *wait-free*. These two properties give rise to a classification of shared variables, depending on their output characteristics. Lamport [8] distinguishes three categories for 1-writer shared variables, using a precedence relation on operation executions defined as follows: for operation executions A and B, A *precedes* B, denoted $A \longrightarrow B$, if A finishes before B starts; A and B *overlap* if neither A precedes B nor B precedes A. In 1-writer variables, all the Writes are totally ordered by "$\longrightarrow$". The three categories of 1-writer shared variables defined by Lamport are the following.

1. A *safe* variable is one in which a Read not overlapping any Write returns the most recently written value. A Read that overlaps a Write may return any value from the domain of the variable.
2. A *regular* variable is a safe variable in which a Read that overlaps one or more Writes returns either the value of the most recent Write preceding the Read or of one of the overlapping Writes.
3. An *atomic* variable is a regular variable in which the Reads and Writes behave as if they occur in some total order which is an extension of the precedence relation.

A shared variable is *boolean*[1] or *multivalued* depending upon whether it can hold only two or more than two values. A *multiwriter* shared variable is one that can be written and read (concurrently) by many processes. If there is only one writer and more than one reader it is called a *multireader* variable.

Key Results

In a series of papers starting in 1974, for details see [4], Lamport explored various notions of concurrent reading and writing of shared variables culminating in the seminal 1986 paper [8]. It formulates the notion of wait-free implementation of an atomic multivalued shared variable—written by a single writer and read by (another) single reader—from safe 1-writer 1-reader 2-valued shared variables, being mathematical versions of physical *flip-flops*, later optimized in [13]. Lamport did not consider constructions of shared variables with more than one writer or reader.

Predating the Lamport paper, in 1983 Peterson [10] published an ingenious wait-free construction of an atomic 1-writer, n-reader m-valued atomic shared variable from $n + 2$ safe 1-writer n-reader m-valued registers, $2n$ 1-writer 1-reader 2-valued atomic shared variables, and 2 1-writer n-reader 2-valued atomic shared variables. He presented also a proper notion of the wait-freedom property. In his paper, Peterson didn't tell how to construct the n-reader boolean atomic variables from flip-flops, while Lamport mentioned the open problem of doing so, and, incidentally, uses a version of Peterson's construction to bridge the algorithmically demanding step from atomic shared bits to atomic shared multivalues. On the basis of this work, N. Lynch, motivated by concurrency control of multi-user data-bases, posed around 1985 the question of how to construct wait-free multiwriter atomic variables from 1-writer multireader atomic variables. Her student Bloom [1] found in 1985 an elegant 2-writer construction, which, however, has resisted generalization to multiwriter. Vitányi and Awerbuch [14] were the first to define and explore the complicated notion of wait-free constructions of general multiwriter atomic variables, in 1986. They presented a proof method, an unbounded solution from 1-writer 1-reader atomic variables, and a bounded solution from 1-writer n-reader atomic variables. The bounded solution turned out not to be atomic,

[1] Boolean variables are referred to as *bits*.

but only achieved regularity ("Errata" in [14]). The paper introduced important notions and techniques in the area, like (bounded) vector clocks, and identified open problems like the construction of atomic wait-free bounded multireader shared variables from flip-flops, and atomic wait-free bounded multiwriter shared variables from the multireader ones. Peterson who had been working on the multiwriter problem for a decade, together with Burns, tried in 1987 to eliminate the error in the unbounded construction of [14] retaining the idea of vector clocks, but replacing the obsolete-information tracking technique by repeated scanning as in [10]. The result [11] was found to be erroneous in the technical report (R. Schaffer, On the correctness of atomic multiwriter registers, Report MIT/LCS/TM-364, 1988). Neither the re-correction in Schaffer's Technical Report, nor the claimed re-correction by the authors of [11] has appeared in print. Also in 1987 there appeared at least five purported solutions for the implementation of 1-writer n-reader atomic shared variable from 1-writer 1-reader ones: [2,7,12] (for the others see [4]) of which [2] was shown to be incorrect (S. Haldar, K. Vidyasankar, *ACM Oper. Syst. Rev*, 26:1(1992), 87–88) and only [12] appeared in journal version. The paper [9], initially a 1987 Harvard Tech Report, resolved all multiuser constructions in one stroke: it constructs a bounded n-writer n-reader (multiwriter) atomic variable from $O(n^2)$ 1-writer 1-reader safe bits, which is optimal, and $O(n^2)$ bit-accesses per Read/Write operation which is optimal as well. It works by making the unbounded solution of [14] bounded, using a new technique, achieving a robust proof of correctness. "Projections" of the construction give specialized constructions for the implementation of 1-writer n-reader (multireader) atomic variables from $O(n^2)$ 1-writer 1-reader ones using $O(^{n})$ bit accesses per Read/Write operation, and for the implementation of n-writer n-reader (multiwriter) atomic variables from n 1-writer n-reader (multireader) ones. The first "projection" is optimal, while the last "projection" may not be optimal since it uses $O(n)$ control bits per writer while only a lower bound of $\Omega(\log n)$ was established. Taking up this challenge, the construction in [6] claims to achieve this lower bound.

Timestamp System

In a multiwriter shared variable it is only required that every process keeps track of which process wrote last. There arises the general question whether every process can keep track of the order of the last Writes by all processes. A. Israeli and M. Li were attracted to the area by the work in [14], and, in an important paper [5], they raised and solved the question of the more general and universally useful notion of a bounded timestamp system to track the order of events in a concurrent system. In a timestamp system every process owns an *object*, an abstraction of a set of shared variables. One of the requirements of the system is to determine the temporal order in which the objects are written. For this purpose, each object is given a *label* (also referred to as a *timestamp*) which indicates the latest (relative) time when it has been written by its owner process. The processes assign labels to their respective objects in such a way that the labels reflect the real-time order in which they are written to. These systems must support two operations, namely *labeling* and *scan*. A labeling operation execution (Labeling, in short) assigns a new label to an object, and a scan operation execution (Scan, in short) enables a process to determine the ordering in which all the objects are written, that is, it returns a set of labeled-objects ordered temporally. The concern is with those systems where operations can be executed *concurrently*, in an overlapped fashion. Moreover, operation executions must be *wait-free*, that is, each operation execution will take a bounded number of its own steps (the number of accesses to the shared space), irrespective of the presence of other operation executions and their relative speeds. Israeli and Li [5] constructed a bit-optimal bounded timestamp system for *sequential* operation executions. Their sequential timestamp system was published in the above journal reference, but the preliminary concurrent timestamp system in the conference proceedings, of which a more detailed version has been circulated in manuscript form, has not been published in final form. The first generally accepted solution of the *concurrent* case of the bounded timestamp system was from Dolev and Shavit [3]. Their construction is of the type presented in [5] and uses shared variables of size $O(n)$, where n is the number of processes in the system. Each Labeling requires $O(n)$ steps, and each Scan $O(n^2 \log n)$ steps. (A 'step' accesses an $O(n)$ bit variable.) In [4] the unbounded construction of [14] is corrected and extended to obtain an efficient version of the more general notion of a bounded concurrent timestamp system.

Applications

Wait-free registers are, together with message-passing systems, the primary interprocess communication method in distributed computing theory. They form the basis of all constructions and protocols, as can be seen in the textbooks. Wait-free constructions of concurrent timestamp systems (CTSs, in short) have been shown to be a powerful tool for solving concurrency control problems such as various types of mutual exclusion, multiwriter multireader shared variables [14], and probabilistic consensus, by synthesizing a "wait-free clock" to sequence the actions in a concurrent system. For more details see [4].

Open Problems

There is a great deal of work in the direction of register constructions that use less constituent parts, or simpler parts, or parts that can tolerate more complex failures, than previous constructions referred to above. Only, of course, if the latter constructions were not yet optimal in the parameter concerned. Further directions are work on wait-free higher-typed objects, as mentioned above, hierarchies of such objects, and probabilistic constructions. This literature is too vast and diverse to be surveyed here.

Experimental Results

Register constructions, or related constructions for asynchronous interprocess communication, are used in current hardware and software.

Cross References

- ▶ Asynchronous Consensus Impossibility
- ▶ Atomic Broadcast
- ▶ Causal Order, Logical Clocks, State Machine Replication
- ▶ Concurrent Programming
- ▶ Emulating Shared-Memory in Message-Passing Systems
- ▶ Linearizability
- ▶ Renaming
- ▶ Self-Stabilization
- ▶ Snapshots in Shared-Memory
- ▶ Synchronizers, Spanners
- ▶ Topology Approach in Distributed Computing
- ▶ Wait-Free Computation

Recommended Reading

1. Bloom, B.: Constructing two-writer atomic registers. IEEE Trans. Comput. **37**(12), 1506–1514 (1988)
2. Burns, J.E., Peterson, G.L.: Constructing multi-reader atomic values from non-atomic values. In: Proc. 6th ACM Symp. Principles Distr. Comput., pp. 222–231. CE2 (1987)
3. Dolev, D., Shavit, N.: Bounded concurrent time-stamp systems are constructible. Siam J. Comput. **26**(2), 418–455 (1997)
4. Haldar, S., Vitanyi, P.: Bounded concurrent timestamp systems using vector clocks. J. Assoc. Comp. Mach. **49**(1), 101–126 (2002)
5. Israeli, A., Li, M.: Bounded time-stamps. Distribut. Comput. **6**, 205–209 (1993) (Preliminary, more extended, version in: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 371–382, 1987.)
6. Israeli, A., Shaham, A.: Optimal multi-writer multireader atomic register. In: Proc. 11th ACM Symp. Principles Distr. Comput., pp. 71–82. CE2 (1992)
7. Kirousis, L.M., Kranakis, E., Vitányi, P.M.B.: Atomic multireader register. In: Proc. Workshop Distributed Algorithms. Lect Notes Comput Sci, vol 312, pp. 278–296. Springer, Berlin (1987)
8. Lamport, L.: On interprocess communication—Part I: Basic formalism, Part II: Algorithms. Distrib. Comput. **1**(2), 77–101 (1986)
9. Li, M., Tromp, J., Vitányi, P.M.B.: How to share concurrent wait-free variables. J. ACM **43**(4), 723–746 (1996) (Preliminary version: Li, M., Vitányi, P.M.B. A very simple construction for atomic multiwriter register. Tech. Rept. TR-01–87, Computer Science Dept., Harvard University, Nov. 1987)
10. Peterson, G.L.: Concurrent reading while writing. ACM Trans. Program. Lang. Syst. **5**(1), 56–65 (1983)
11. Peterson, G.L., Burns, J.E.: Concurrent reading while writing II: The multiwriter case. In: Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 383–392. CE2 (1987)
12. Singh, A.K., Anderson, J.H., Gouda, M.G.: The elusive atomic register. J. ACM **41**(2), 311–339 (1994) (Preliminary version in: Proc. 6th ACM Symp. Principles Distribt. Comput., 1987)
13. Tromp, J.: How to construct an atomic variable. In: Proc. Workshop Distrib. Algorithms. Lecture Notes in Computer Science, vol. 392, pp. 292–302. Springer, Berlin (1989)
14. Vitányi, P.M.B., Awerbuch, B.: Atomic shared register access by asynchronous hardware. In: Proc. 27th IEEE Symp. Found. Comput. Sci. pp. 233–243. Errata, Proc. 28th IEEE Symp. Found. Comput. Sci., pp. 487–487. CE2 (1986)