Brief Announcement: How Fast Reads Affect Multi-Valued Register Simulations

Soma Chaudhuri
Iowa State University
chaudhur@iastate.edu

Reginald Frank
Texas A&M University
reginaldfrank77@tamu.edu

Jennifer L. Welch
Texas A&M University
welch@cse.tamu.edu

ABSTRACT

We consider the problem of simulating a k-valued register in a wait-free manner using binary registers as building blocks, where $k > 2$. We show that for any simulation using atomic binary base registers to simulate a safe k-valued register in which the read algorithm takes the optimal number of steps ($\log_2 k$), the write algorithm must take at least $\log_2 k$ steps in the worst case. *A fortiori*, the same lower bound applies when the simulated register should be regular. Previously known algorithms show that both these lower bounds are tight. We also show that in order to simulate an atomic k-valued register for two readers, the optimal number of steps for the read algorithm must be strictly larger than $\log_2 k$.

CCS CONCEPTS

• **Theory of Computation → Concurrency**.

KEYWORDS

register simulations, wait-freedom

ACM Reference Format:
Soma Chaudhuri, Reginald Frank, and Jennifer L. Welch. 2019. Brief Announcement: How Fast Reads Affect Multi-Valued Register Simulations. In *2019 ACM Symposium on Principles of Distributed Computing (PODC '19), July 29-August 2, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3293611.3331580

1 INTRODUCTION

We consider the problem of how to use a collection of shared read-write registers with certain properties to simulate a shared read-write register with stronger properties in a wait-free manner. Each simulated read or write operation can access some of the building block, or "base", registers. In a *wait-free* simulation, each simulated operation terminates in a finite number of steps by the invoking process, regardless of the behavior of the other processes, which may be arbitrarily fast, or arbitrarily slow, or even crash. Lamport formalized the notions of safe, regular, and atomic registers, proposed a formal model, and provided numerous algorithms for simulating certain kinds of registers out of other kinds [6].

Simulating a k-valued register using binary base registers, where $k > 2$, has been studied in, e.g., [3, 4, 6, 8], while generalizations for simulating large registers out of small registers appear in [1, 2, 7].

PODC '19, July 29-August 2, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6217-7/19/07.
https://doi.org/10.1145/3293611.3331580

Regardless of the consistency conditions of the simulated register and the base registers, the number of steps taken by the simulated read must be at least $\log_2 k$ [4]. However, little work has been done in exploring tradeoff results in the optimal number of base registers or the optimal number of steps taken by the simulated write when the simulated read is limited to $\log_2 k$ steps. The only previous work we are aware of is an asymptotic result that, for the regular case, if the simulated read takes $O(\log k)$ steps, then the simulated write must take $\Omega(\log k)$ steps [5].

In this paper we consider what happens when the simulated read is limited to $\log_2 k$ steps. In the safe and regular cases, we show that this restriction results in a lower bound of $\log_2 k$ on the number of steps that the simulated write must take. These lower bounds are tight due to algorithms in [6] for the safe case and [4] for the regular case. In the atomic case we show that, if there are at least two readers, no such algorithm is possible, thus giving a lower bound of $1 + \log_2 k$ on the number of steps by the simulated read. This bound is not tight as the best known upper bound is $1 + 2 \cdot \log_2 k$ [5].

2 MODEL AND DEFINITIONS

The goal is to simulate a k-valued register $\mathbb{X}$, with value set V, $|V| = k > 2$, that supports n readers and one writer. We assume k is a power of 2. When the i-th reader invokes a Read on $\mathbb{X}$, a read process, denoted RP_i, $1 \leq i \leq n$, performs some computation and eventually outputs a Return with a value in V to the reader. Similarly, when the writer invokes a Write with a value in V on $\mathbb{X}$, the write process, denoted WP, performs some computation and eventually outputs an Ack to the writer. The read and write processes communicate with other through a finite set $\mathbb{B}$ of atomic base registers, each with a binary value set. Since the base registers are atomic, we model the read and write operations on them as occurring instantaneously. The values appearing in the Returns should satisfy one of the following consistency conditions [6]: *safety* (each simulated read that does not overlap a simulated write must return the value of the latest preceding simulated write), *regularity* (each simulated read must return the value of an overlapping simulated write or of the latest preceding simulated write), or *atomicity* (it should appear as if each simulated operation takes place instantaneously at some point between its invocation and response). Furthermore, as long as any particular process keeps taking steps, it should provide responses to the invocations on $\mathbb{X}$ no matter how the other processes are scheduled.

All bounds are worst-case.

3 SAFE AND REGULAR: STEP LOWER BOUND FOR WRITE WHEN READ IS FAST

An easy extension to a result in [4] shows:

Theorem 3.1. *For any wait-free simulation of a safe k-valued register using atomic binary registers, the number of steps in the simulated read is at least $\log_2 k$.*

The main result of the section is next.

Theorem 3.2. *For any wait-free simulation of a safe k-valued register using atomic binary registers, if the simulated read uses at most $\log_2 k$ reads, then the simulated write uses at least $\log_2 k$ writes.*

Proof. Let $\mathbb{A}$ be any such simulation and V be the value set of the simulated register, $|V| = k$.

For each i, $1 \leq i \leq n$, we recursively define the *decision tree* of read process RP_i, denoted T_i, as follows. The purpose is to capture which base registers are read during the first simulated read that RP_i executes in any execution. Each vertex in T_i is labeled with either a base register or a value in V. The basis step defines the root of T_i as a vertex labeled with the first base register read in the first simulated read performed by RP_i. For the inductive step, let x be a vertex in T_i labeled with register r. If the code tells RP_i to read base register s next after reading 0 (resp., 1) from r, then give x a vertex as its left (resp., right) child labeled with s. If the code tells RP_i to do Return(v) next after reading 0 (resp., 1) from r, then give x a vertex as its left (resp., right) child labeled with v. Continue until all leaves of T_i are labeled with values.

For each $v \in V$ and each i, $1 \leq i \leq n$, define σ_v^i as the execution of $\mathbb{A}$ in which a simulated write of v is performed by WP followed by a simulated read by RP_i. By correctness (i.e., safety) of $\mathbb{A}$, the simulated read must return v.

Lemma 3.3. *For each i, $1 \leq i \leq n$, T_i is a complete binary tree with exactly k leaves in which every leaf is at depth $\log_2 k$ and every root-to-leaf path π corresponds to the sequence of base registers read during the simulated read in σ_v^i, where v is is the label of the leaf in π.*

Proof. Fix i, $1 \leq i \leq n$. The existence of σ_v^i for each $v \in V$ implies that T_i must have at least k leaves. By the assumption that the step complexity of the simulated read is at most $\log_2 k$, every root-to-leaf path in T_i has at most $\log_2 k$ internal vertices. Since T_i is a binary tree, basic facts from graph theory imply that T_i has exactly k leaves and each leaf is at depth exactly $\log_2 k$. Thus the root-to-leaf path π ending at the leaf labeled v corresponds to the sequence of base registers read during the simulated read in σ_v^i. □

Note that the first simulated read by each RP_i is invisible, i.e., contains no writes to base registers, since it must have $\log_2 k$ reads in it and we assume that the step complexity is at most $\log_2 k$.

Lemma 3.4. *Let π be any root-to-leaf path in T_i, for any i, $1 \leq i \leq n$. Then every internal node on π has a unique register label.*

Proof. Fix i, $1 \leq i \leq n$. Suppose in contradiction some root-to-leaf path π has two occurrences of the register label r, say at distances a and b from the root, with $a < b$. The tree vertex representing the b-th read in π has two children, since T_i is complete by Lemma 3.3, one of which is on π. Let π' be a root-to-leaf path in T_i that is the same as π through the b-th read but then diverges. By Lemma 3.3, π corresponds to execution σ_v^i for some $v \in V$ and π' corresponds to execution $\sigma_{v'}^i$ for some $v' \in V$ with $v' \neq v$.

However, it is not possible for σ_v^i and $\sigma_{v'}^i$ to read different values from r at the b-th read since the only process that is taking steps during the simulated reads is RP_i: even if RP_i writes to r during the simulated read, it will write the same value in σ_v^i as in $\sigma_{v'}^i$, as nothing differs in those two executions until reaching the b-th read. □

Now we can finish the proof of the theorem. Without loss of generality, assume that the initial value of every base register is 0. Consider the rightmost root-to-leaf path in the decision tree T_1. By Lemma 3.3, in the corresponding execution σ_v^1, the reader RP_1 reads 1 from a base register $\log_2 k$ times during the simulated read. By Lemma 3.4, none of the base registers read during the simulated read is a repeat. Thus during the simulated write in σ_v^1, the writer WP writes 1 to the each of the $\log_2 k$ base registers that are read during the simulated read. □

Since regularity is stronger than safety, we have the following corollary to the preceding theorem:

Corollary 3.5. *For any wait-free simulation of a regular k-valued register using atomic binary registers, if the simulated read uses at most $\log_2 k$ reads, then the simulated write uses at least $\log_2 k$ writes.*

4 ATOMIC: STEP LOWER BOUND FOR READ

Now we consider the atomic case. We show that when there are two read processes, then it is impossible to simulate a k-valued register if the simulated read takes only $\log_2 k$ steps. The key to the proof is showing that, if the simulated read takes only $\log_2 k$ steps, then the first simulated read by each of the read processes must start by reading the same binary register first. We use the concept of decision tree and Lemmas 3.3 and 3.4 from inside the proof of Theorem 3.1.

Theorem 4.1. *No wait-free simulation of an atomic k-valued register using atomic binary registers can have a simulated read that takes at most $\log_2 k$ steps, assuming there are at least two readers.*

Proof. Assume in contradiction there is such a simulation $\mathbb{A}$, with readers RP_1 and RP_2. We already observed that their first simulated reads are invisible (contain no writes to base registers). Let T_i be the decision tree of the first simulated read by RP_i, $i = 1, 2$.

Lemma 4.2. *T_1 and T_2 have the same root register.*

Proof. Suppose in contradiction the root r_1 of T_1 is not equal to the root r_2 of T_2. Let z be a value whose root-to-leaf path in T_2 does not include r_1. Such a value must exist since r_1 is not the root of T_2. Without loss of generality, suppose z appears in the right half of the leaf level of T_1. Let a be a value that appears in the left half of the leaf level of T_1.

Consider an execution e consisting of two simulated writes, $W_0(a)$ followed by $W_1(z)$. After W_0, r_1 must equal 0, as it's possible that RP_1's first simulated read is immediately after W_0, in which case it must return a and thus must see 0 in r_1. A similar argument shows that after W_1, r_1 must equal 1. Thus W_1 must contain a write of 1 to r_1.

Consider adding two simulated reads to e to make execution e': immediately after the first write of 1 to r_1 in W_1, insert RP_1's first simulated read R_1, immediately followed by RP_2's first simulated

read R_2. By regularity, R_1 must return either a or z and since R_1 observes 1 in r_1, it must return z. Then by atomicity, R_2 must return z as well.

Consider execution e'' which is obtained from e' by moving R_1 to appear immediately before the first write of 1 to r_1 in W_1 and moving R_2 to appear immediately before R_1. The only difference in the registers during the execution of R_2 in e'' versus e' is the value of r_1. But r_1 does not lie on the root-to-leaf path in T_2 leading to z, and thus R_2 will behave the same in e'' as it does in e' and return z.

Consider the occurrence of R_1 in e''. By atomicity, R_1 must return z, but R_1 reads 0 in r_1 and thus cannot return z, a contradiction. □

By Lemma 4.2, T_1 and T_2 have the same root, call it register r. Let a and b be two "sibling" values that are in the left half of the leaf level of T_1, with a to the left of b, and z be a value that is in the right half of the leaf level of T_1. Since T_1 and T_2 have the same root r, a and b are also in the left half of the leaf level of T_2 and z is also in the right half of the leaf level of T_2. Let x be the register that is the parent of a and b in T_1.

Consider an execution e consisting of three simulated writes: $W_0(a)$ followed by $W_1(z)$, followed by $W_2(b)$. We make the following observations:

- After W_0 finishes, $r = x = 0$. The reason is that it is possible that RP_1 does its first simulatd read between W_0 and W_1, which must return a by safety and thus T_1 must have $r = x = 0$.
- After W_1 finishes, $r = 1$. The reason is that it is possible that RP_1 does its first simulated read between W_1 and W_2, which must return z by safety and thus T_1 must have $r = 1$. So during W_1 there must be at least one write of 1 to r.
- After W_2 finishes, $r = 0$ and $x = 1$. The reason is that it is possible that RP_1 does its first simulated read after W_2, which must return b by safety and thus T_1 must have $r = 0$ and $x = 1$. So during W_2 there must be at least one write of 0 to r during W_2. Also, there must be at least one write of 1 to x in the interval between the beginning of W_1 and the end of W_2.

The rest of the proof has two cases, depending on whether the first write of 1 to x between the beginning of W_1 and the end of W_2 occurs before or after the first write of 0 to r in W_2; if before, we will show that atomicity is violated; if after, that regularity is violated.

Case 1: The first write of 1 to x occurs before the first write of 0 to r in W_2.

Now consider another execution e_1 in which we add a simulated read R_1 by RP_1 and a simulated read R_2 by RP_2 to execution e. Since simulated reads are invisible, their existence does not affect the behavior of the simulated writes in e.

Suppose R_1 starts after W_0 ends and reads r through the parent of x in T_1 before W_1 begins. Then R_1 reads x immediately after the first write of 1 to x (which occurs in either W_1 or W_2), and thus sees $x = 1$. So R_1 returns b, as indicated by its decision tree T_1.

Now suppose R_2 starts after R_1 ends and reads r before W_2's first write of 0 to r, and thus sees $r = 1$. By definition of atomicity, the only legal value that R_2 can return is b, but by the structure of its decision tree T_2 it will not return b, since it read 1 from r. Contradiction.

Case 2: The first write of 1 to x occurs after the first write of 0 to r in W_2.

Now consider another execution e_2 in which we add a simulated read R_1 by RP_1 to execution e. As in Case 1, since simulated reads are invisible, its existence does not affect the behavior of the simulated writes in e.

Suppose R_1 starts and reads r immediately after the first write of 0 to r in W_2, seeing $r = 0$, and then R_1 completes all its reads before the first write of 1 to x in W_2. The structure of the decision tree T_1 implies that R_1 cannot return z, since R_1 read 0 in r, and R_1 cannot return b, since R_1 read 0 in x. But b and z are the only valid return values that satisfy atomicity (or even regularity for that matter). Contradiction. □

The same impossibility holds with only one read process assuming that its first two simulated reads start by reading the same binary register.

5 CONCLUSION

We have shown how having a fast read algorithm affects the speed of the write algorithm when simulating a safe or regular k-valued register and that the read algorithm must be slower in the atomic case than in the safe and regular cases. Open questions include getting rid of the need for two readers to show the $1 + \log_2 k$ lower bound on the read algorithm step complexity in the atomic case. More generally, we would like to close the gap between the upper bound of $1 + 2 \cdot \log_2 k$ on this measure due to [5] and the lower bound.

ACKNOWLEDGMENTS

This work was supported in part by NSF grant 1816922.

REFERENCES

[1] Zahra Aghazadeh, Wojciech M. Golab, and Philipp Woelfel. Making objects writable. In *ACM Symposium on Principles of Distributed Computing*, pages 385–395, 2014.
[2] Alon Berger, Idit Keidar, and Alexander Spiegelman. Integrated bounds for disintegrated storage. In *32nd International Symposium on Distributed Computing*, pages 11:1–11:18, 2018.
[3] Soma Chaudhuri, Martha J. Kosa, and Jennifer L. Welch. One-write algorithms for multivalued regular and atomic registers. *Acta Inf.*, 37(3):161–192, 2000.
[4] Soma Chaudhuri and Jennifer L. Welch. Bounds on the costs of multivalued register implementations. *SIAM J. Comput.*, 23(2):335–354, 1994.
[5] Tian Ze Chen and Yuanhao Wei. Step optimal implementations of large single-writer registers. In *20th International Conference on Principles of Distributed Systems (OPODIS)*, pages 32:1–32:16, 2016.
[6] Leslie Lamport. On interprocess communication. Part II: Algorithms. *Distributed Computing*, 1(2):86–101, 1986.
[7] Gary L. Peterson. Concurrent reading while writing. *ACM Trans. Program. Lang. Syst.*, 5(1):46–55, 1983.
[8] K. Vidyasankar. Converting Lamport's regular register to atomic register. *Inf. Process. Lett.*, 28(6):287–290, 1988.