Regular Tree Algorithm Redux: The Atomic Case: DREU Final Report

Soma Chaudhuri
Iowa State University, USA
chaudur@iastate.edu

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

Jennifer L. Welch[1]
Texas A&M University, USA
welch@cse.tamu.edu

Abstract

We consider the problem of simulating a k-valued atomic register in a wait-free manner using binary atomic registers as building blocks, for any $k > 2$. In this note we prove that the tree algorithm of Chaudhuri and Welch [1], which was originally proposed to simulate a k-valued *regular* register out of binary *regular* registers, simulates an *atomic* register when the binary registers are *atomic*. Our tree algorithm uses $k - 1$ binary registers and each operation on the simulated register performs $\lceil \log_2 k \rceil$ operations on the binary registers. When the number of readers of the simulated register is $\omega\left(\frac{k}{\log k}\right)$, this is the most space-efficient algorithm known for the problem. Furthermore we prove that the said algorithm's worst case logical WRITE use the fewest amount of physical writes possible.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Interprocess communication, Shared registers

1 Introduction

A *register* is a fundamental shared object which can either be read from or written to. Our work focuses on registers with a single writer, but an arbitrary number of readers. Different registers may have different properties such as *regularity* or *atomicity*. *Regularity* ensures that if a read of the register is concurrent with a write, then the read will return either the previous (if no preceding writes, initial) value of the register or the value of the concurrent write; otherwise the read will return the most recent value held/written to the register. *Atomicity* is stronger than regularity as it provides a definite order to the operations in an execution, meaning the operations are *linearizable* [4]. Furthermore it is accepted [3] that the only distinction between regular and atomic executions is that regular executions allow for *new−old* inversions. These inversions occur when one or more read returns an *old* value, the value which was written just before the current write after the current write has been read. Reads of a regular register are permitted to return old values allowing for *new−old* inversions, however atomicity forbids this.

Several algorithms have been created to make use of a number of binary registers for the purpose of constructing k-valued registers, a register which can hold k unique values where $k \in \mathrm{Z}^+$. In particular, the tree algorithm created by Chaudhuri and Welch in [1] uses $k - 1$

[1] Supported in part by NSF grant 1526725.

binary regular registers to construct a k-valued regular register. The motive of this document is to show that this algorithm can directly be extended to use atomic binary registers to create an atomic k-valued register. This will be done by showing that "new-old inversions" are not possible in the resulting construction.

2 Tree Algorithm

The original tree algorithm presented in [1] goes as follows. For a value set V of size k, associate a binary register with each internal node of a binary tree with k leaves and label the leaves with the elements of V. A READ operation reads the registers corresponding to a path from the root down to a leaf. At each level, if the current register holds 0 (resp., 1), then the next register read is the left (resp., right) child. If the last register in the path holds 0 (resp., 1), then return the value labeling the left (resp., right) child of the tree node corresponding to the register. A WRITE(v) operation writes the registers corresponding to the path from the leaf labeled v up to the root. At each level, if the previous tree node is the left (resp., right) child of the current tree node, then 0 (resp., 1) is written to the register corresponding to the current tree node. The tree algorithm uses $k-1$ binary registers. If the binary tree is complete, then the number of steps in each operation is at most $\lceil \log_2 k \rceil$.

It is shown in [1] that if the binary registers are regular, then the k-valued register simulated by the tree algorithm is also regular. We will show that if the binary registers are atomic, then the tree algorithm simulates an atomic register.

To simplify the correctness proof, we provide a new way of describing the tree algorithm, called $RTA(V)$ for Recursive Tree Algorithm, where V is the finite set of ordered values to be stored in the register. For now, assume that $|V| = k$ is a power of 2. (Later we will discuss how to relax this assumption.)

For the base case, when $|V| = 2$, $RTA(V)$ uses a single binary[2] atomic register x, the READ algorithm consists solely of reading x and returning the value obtained, and the WRITE(v) algorithm consists solely of writing v to x.

Now suppose $|V| > 2$. $RTA(V)$ can be viewed as a binary tree with two levels. The root of the tree is a binary atomic register, *root*, with two children. The left child, denoted as **A**, and the right child, denoted as **B**, are both $\frac{k}{2}$-valued atomic registers, where **A**'s value set consists of the $\frac{k}{2}$ smallest values in V and **B**'s value set consists of the $\frac{k}{2}$ largest values in V.

The READ algorithm first reads the root; if 0 (resp., 1) is returned, then it reads **A** (resp., **B**) and returns the value obtained. Pseudocode is in Algorithm 1. To WRITE v when v is less than the median of V, the algorithm first writes v to **A** and then writes 0 to the root; when v is greater than the median of V, it first writes v to **B** and then writes 1 to the root. Pseudocode is in Algorithm 2.

[2] The size of the value set is 2, but the values are not necessarily 0 and 1.

Algorithm 1 *RTA* READ, $|V| > 2$

```
1: procedure READ()
2:     rootVal ← read (root)
3:     if rootVal = 0 then
4:         RETURN read(A)
5:     else
6:         RETURN read(B)
```

Algorithm 2 *RTA* WRITE, $|V| > 2$

```
1: procedure WRITE(v)
2:     if v < median(V) then
3:         write(v) to A
4:         write(0) to root
5:     else
6:         write(v) to B
7:         write(1) to root
8:     ACK
```

To simulate the k-valued register from binary registers, use *RTA* recursively in this construction. That is, let $\mathbf{A} = RTA(V_1)$, where V_1 consists of the $\frac{k}{2}$ smallest elements of V and let $\mathbf{B} = RTA(V_2)$, where V_2 consists of the $\frac{k}{2}$ largest elements of V.

When the recursion is unrolled, the resulting algorithm is equivalent to the tree algorithm in [1]. Informally the reason is that all key features of the tree algorithm, such as WRITEs progressing from leaf to root, and READs progressing from root to leaf, are preserved.

3 Analysis

In this section, we show that $RTA(V)$ simulates an atomic register.

For an execution of an atomic register x, we say that read r of x *reads from* a write w of x if w is the latest write that precedes r in the linearization of the operations on x.

▶ **Theorem 1.** *$RTA(V)$ implements an atomic register when $|V| = k$ is a power of 2.*

Proof. We use induction on the size of V to show that $RTA(V)$ is atomic.

Base Case: $|V| = 2$. $RTA(V)$ is simply a binary atomic register and the result follows.

Inductive Step: $|V| = 2^m$, where $m > 1$. By the inductive hypothesis, the building block registers $\mathbf{A}$ and $\mathbf{B}$ are atomic since each of them is an instantiation of *RTA* for a value set of size 2^{m-1}.

By a result of Lamport [4], to show that an execution is atomic, it is sufficient to demonstrate the existence of a function ρ from the set of READs in the execution to the set of WRITEs in the execution such that:

1. For every READ R, $\rho(R)$ either overlaps R or is the latest WRITE that finishes before R begins.
2. For all READs R_1 and R_2 such that R_1 finishes before R_2 begins, $\rho(R_1)$ either equals or precedes $\rho(R_2)$.

The first condition ensures regularity and the second condition rules out "new-old" inversions.

For the tree algorithm, we define the function ρ as follows. For READ R, let $\rho(R)$ be the WRITE W such that R's leaf read (of $\mathbf{A}$ or $\mathbf{B}$) reads from W's leaf write. Note that the value written to the leaf is the same as the value read from that leaf, and thus the value returned by R is the same as the value written by W.

It is straightforward to see that ρ satisfies condition (1). Suppose for contradiction that ρ does not satisfy condition (2). Then there are two READs, $R_1(v_1)$ and $R_2(v_2)$, such that R_1 finishes before R_2 begins but $\rho(R_2) = W_2(v_2)$ finishes before $\rho(R_1) = W_1(v_1)$ begins.

Case 1: R_1 and R_2 read the same value from *root*, without loss of generality, say 0. This means they both read leaf $\mathbf{A}$, resulting in the following precedence relations (denoted $\rightarrow$); see Figure 1.

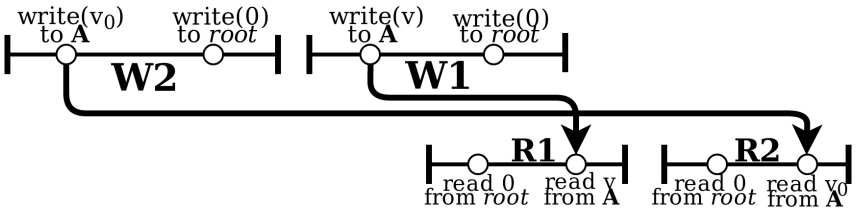

Figure 1 Precedence relations for Case 1

- read(0) from *root* in $R_1 \rightarrow$ read(v_1) from **A** in $R_1 \rightarrow$ read(0) from *root* in $R_2 \rightarrow$ read(v_2) from **A** in R_2
- write(v_2) to **A** in $W_2 \rightarrow$ write(0) to *root* in $W_2 \rightarrow$ write(v_1) to **A** in $W_1 \rightarrow$ write(0) to *root* in W_1
- write(v_1) to **A** in $W_1 \rightarrow$ read(v_1) from **A** in R_1
- write(v_2) to **A** in $W_2 \rightarrow$ read(v_2) from **A** in R_2

Since W_2's write to **A** is followed by W_1's write to **A**, which is followed by R_1's read of **A**, which is followed by R_2's read of **A**, it is not possible for $\rho(R_2)$ to be W_2, a contradiction.

Case 2: R_1 and R_2 read different values from *root*; without loss of generality suppose R_1 reads 0 and thus reads leaf **A**, while R_2 reads 1 and thus reads leaf **B**.

We now have the following precedence relations:

- read(0) from *root* in $R_1 \rightarrow$ read(v_1) from **A** in $R_1 \rightarrow$ read(1) from *root* in $R_2 \rightarrow$ read(v_2) from **B** in R_2
- write(v_2) to **B** in $W_2 \rightarrow$ write(1) to *root* in $W_2 \rightarrow$ write(v_1) to **A** in $W_1 \rightarrow$ write(0) to *root* in W_1
- write(v_1) to **A** in $W_1 \rightarrow$ read(v_1) from **A** in R_1
- write(v_2) to **B** in $W_2 \rightarrow$ read(v_2) from **B** in R_2

One important observation is that if a WRITE writes 0 (resp., 1) to *root* then it has previously written to leaf **A** (resp., **B**).

Case 2.1: R_1's read of *root* reads from W_1's write to *root* or a later one. See Figure 2. Then R_2's read of *root* reads from a write to *root* in W_1 or a later WRITE. By the observation, R_2's read of **B** reads from the write to **B** in W_1 or a later WRITE, so $\rho(R_2)$ cannot be W_2, a contradiction.

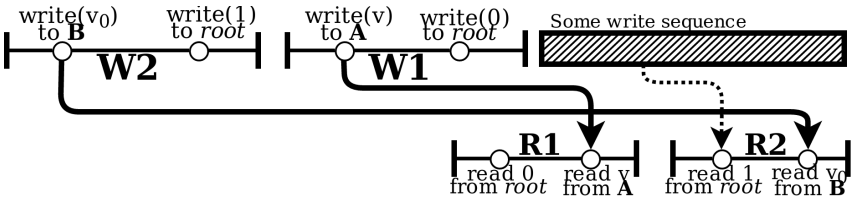

Figure 2 Precedence relations for Case 2.1

Case 2.2: R_1's read of *root* reads from the write to *root* in a WRITE earlier than W_1. It cannot be from W_2, since W_2 writes a different value to *root* than what R_1 reads. It cannot be from a WRITE that precedes W_2, since earlier writes to *root* are followed by W_2's. So it must be from some WRITE W_3 that is in between W_2 and W_1. See Figure 3. But then W_2's write to *root* precedes W_3's write to *root*, which precedes R_1's read of *root*, which precedes R_2's read of *root*. Thus R_2's read of *root* reads from a write to *root* in some WRITE W_4 that

follows W_3. By the observation, W_4 writes **B** before it writes *root*, and thus $\pi(R_2)$ cannot be W_2, a contradiction.

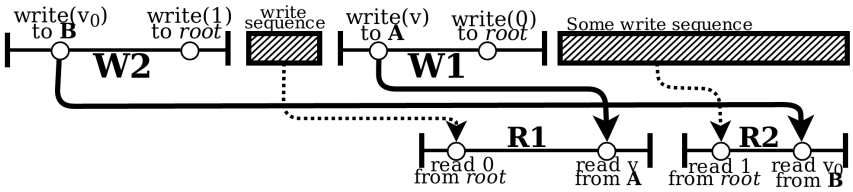

Figure 3 Precedence relations for Case 2.2

Thus ρ satisfies condition 2 and hence $RTA(V)$ is atomic. ◀

As noted in [1] for the original tree algorithm, $RTA(V)$ uses $k-1$ binary registers, where $|V| = k$, and the number of steps per operation is $\log_2 k$.

4 Extensions

Because *RTA* is equivalent to the Tree Algorithm with atomic binary registers the same method can be used to allow for a value of k which is not a power of 2. The general intuition for this is that each implementation where k is not a power of two, RTA provides a tree which is equivalent to a tree where k is the following power of 2 with a number of nodes removed. The restricted nodes has an equivalent purpose as restricting WRITEs to only use values between 0 and k; it follows that the all the possible executions of this restricted implementation are a subset of the set of possible executions when WRITEs are not restricted. Because the unrestricted set of executions are all linearizable, it follows then that the restricted set is also linearizable showing that the original restriction on k is not necessary to implement an atomic *RTA* algorithm.

This shows that RTA can correctly implement an atomic register for any sized k, allowing **A** and **B** to be created using RTA meaning all implementations of RTA can be constructed solely using binary atomic registers.

5 Lower Bound

In this section we give several facts about algorithms where all logical READs use $\leq log_2(k)$ physical reads.

▶ **Theorem 2.** *For any wait-free algorithm A that simulates a k-valued safe register using binary atomic registers, if the* READ *algorithm uses at most $log_2(k)$ reads in the worst case, then the* WRITE *algorithm uses at least $log_2(k)$ writes in the worst case.*

Proof. Consider the following set S of executions of A: σ_i = WRITE(v_i) α_i ACK READ$_1$ β_i RETURN(w_i) | only the WP takes steps in α_i and only RP_1 takes steps in β_i, $0 \leq i < k$. Note that w_i must equal v_i to satisfy safety.

Now construct a decision tree of RP_1's behavior in the executions in S. The root of the decision tree corresponds to the first register that RP_1 reads, this register is the same for all executions in S as RP_1 always begins in the same state. Now depending on the first value read, RP_1 does some amount of non-read actions then does another read of some register. Because the root only can store two values there are only two registers RP_1 can decide to read from causing the root to have two children. Continue to build the tree like this. Add leaves to indicate which value RP_1 will RETURN.

▶ **Lemma 3.** *The decision tree is a complete binary tree with exactly k leaves in which every leaf is at depth $log_2(k)$ and the path from the root to the leaf labeled v_i corresponds to $\beta_i \in \sigma_i$.*

Proof. Note that there must be at least k leaves in the decision tree, as each value in V is returned in one of the executions in S. Since the decision tree is binary, basic facts from graph theory imply that each leaf must be at depth $log_2(k)$, allowing exactly k leaves and each root-to-leaf path must correspond to a different execution in S. ◀

▶ **Lemma 4.** *None of the* Reads *in an execution in S reads the same register more than once.*

Proof. Suppose in contradiction the Read in σ_i reads some number of registers more than once, for some i. Let x be the first register that RP_1 reads twice, say the a-th read and the b-th read, with $log_2(k) \geq b > a$. Consider the node in the decision tree for the b-th read in the path for σ_i. This node must have two children since the decision tree is complete. Consider a root-to-leaf path π in the decision tree that forks off from the path corresponding to σ_i at this node. By Lemma 1, there is a one-to-one correspondence between root-to-leaf paths in the decision tree and the set of executions and so π corresponds to σ_j in S, for some $j \neq i$.

However, it is not possible for σ_i and σ_j to read different values from x at the b-th read since the only process that is taking steps during the Reads is RP_1: even if RP_1 writes to x during the Read, it will write the same value in σ_i as in σ_j, as nothing differs in those two executions until reaching the b-th read. ◀

Now we can finish the proof of the theorem. Let x_1 be the first register read in the Read of each execution in S. Note that in half of the elements of S, the value read from x_1 in the Read must be different from the initial value of x_1, in order to be able to reach all the leaves on that half of the decision tree. In other words, the WRITE writes to x_1 in half the executions in S. Let S_1 be the subset of S consisting of the executions in which the Write writes x_1; note that $|S_1| = \frac{k}{2}$. Now let x_2 be the register corresponding to the root of subtree in which x_1 was written. A similar argument shows that in half the executions in S_1, the value read from x_2 in the Read must be different from the initial value of x_2. Let S_2 be the subset of S_1 consisting of the executions in which the Write writes to x_2; note that $|S_2| = \frac{k}{2^2}$. Continuing this way we obtain $S_{log_2 k}$, which is of size 1, in which the Write writes to $x_1, x_2, ..., x_{log_2 x}$. By Lemma 2, each of these registers is distinct and thus the WRITE writes to at least $log_2(k)$ registers. ◀

6 Conclusion

This result directly improves on the Tree-Algorithm found in [2], however it does not improve their combined implementation which uses the un-optimal tree algorithm. And in conclusion the tree algorithm uses an optimal number steps in both Reads and Writes when all Reads use $\leq log_2(k)$ physical reads.

References

1 Soma Chaudhuri and Jennifer L. Welch. Bounds on the costs of multivalued register implementations. *SIAM J. Comput.*, 23(2):335–354, 1994.

217 **2** Tian Ze Chen and Yuanhao Wei. Step optimal implementations of large single-writer
218 registers. In *20th International Conference on Principles of Distributed Systems (OPODIS)*,
219 pages 32:1–32:16, 2016.
220 **3** Rachid Guerraoui and Petr Kuznetsov. *Concurrent Computing*, January 21, 2018 (accessed
221 July 4, 2018). https://perso.telecom-paristech.fr/kuznetso/MITRO207-2018/book-ln.pdf.
222 **4** Leslie Lamport. On interprocess communication. Part II: Algorithms. *Distributed Comput-*
223 *ing*, 1(2):86–101, 1986.