There Are No Doubly Non-Interleaving List CRDTs

Matthew Weidner[1]

[1] *Carnegie Mellon University, maweidne@andrew.cmu.edu*

December 11, 2021

1 Introduction

Conflict-free Replicated Data Types (CRDTs) [13, 10] are highly available replicated data types used in distributed key-value stores [2] and collaborative web apps [8]. A *list CRDT* is a CRDT representing a list, with operations to insert and delete elements. A prominent use case for list CRDTs is collaborative text editing: represent the text as a list of characters.

Numerous list CRDTs have been proposed, e.g., [11, 12, 14, 9]. For collaborative text editing, though, we typically want to restrict to CRDTs that are *non-interleaving*: if two users type a series of characters at the same location concurrently, then their edits should appear one after the other, not be interleaved. See Kleppmann et al. [7] for details.

RGA (Replicated Growable Array) [11] is one list CRDT. It satisfies a reasonable non-interleaving property for LtR insertions: if two users both type a series of characters *from left to right* at the same location concurrently, then their edits will appear one after the other [6, 5]. However, RGA does not satisfy the analogous non-interleaving property for RtL insertions [7]. Kleppmann et al. proposed a variant of RGA and conjectured that it satisfied both LtR and RtL non-interleaving, but their CRDT was incorrect [4].

In this paper, we show that for fairly weak definitions of LtR and RtL non-interleaving, no text CRDT can satisfy both simultaneously. Thus at least with respect to these definitions, there are no "doubly non-interleaving" list CRDTs. To prove this, we first show that our seemingly weak definition of LtR non-interleaving is equivalent to a much stronger version, and any algorithm satisfying it must be similar to RGA (Theorem 2.2)—an interesting result in its own right. We then give a counterexample to double non-interleaving in the form of a text editing trace (Theorem 3.1).

Finally, although a doubly non-interleaving list CRDT is impossible, we show that one can get close using a novel list CRDT we call Double RGA.

2 From Non-Interleaving to an RGA-Like Algorithm

In a text CRDT, the *left origin* of an element is the element directly to its left at the time of insertion. The left origin relation gives a tree structure on elements, rooted at a special start element. We will call this tree the *left origin tree*; previous works have called it a *timestamped insertion tree* [1] or *causal tree* [3].

Assume a list CRDT has the following non-interleaving property for LtR insertions:

Definition 2.1 (LtR Non-Interleaving)**.** Suppose list elements a and $b_1, \ldots, b_k$ satisfy:

- a and b_1 have the same left origin.
- $b_1, \ldots, b_k$ form a chain of left origins, i.e., the left origin of b_i is b_{i-1} for all $i \geq 2$.
- a is concurrent to all $b_1, \ldots, b_k$.

Then in the final list order, all b_i are on the same side of a, i.e., either $a < b_1, \ldots, b_k$ or $b_1, \ldots, b_k < a$.

In other words, if an element a is concurrent to some elements $b_1, \ldots, b_k$ inserted in sequence from left to right, then a must not be interleaved in the middle of the sequence.

This is a "minimal" definition of LtR non-interleaving. At first glance, it appears weaker than more useful user-facing notions, like a requirement that multiple concurrent LtR sequences should not be interleaved. However, it turns out that it is already sufficient to imply something much stronger, namely, an RGA-style algorithm:

Theorem 2.2. *Assuming Definition 2.1, the final list order is a tree walk over the left origin tree in which each element is ordered before its children, for some ordering of siblings in the tree.*

Corollary 2.3. *Assuming Definition 2.1, suppose list elements $a_1, \ldots, a_l$ and $b_1, \ldots b_k$ are such that a_1 and b_1 have the same left origin, and $a_1, \ldots, a_l$ and $b_1, \ldots, b_k$ each form a chain of left origins. Then in the final list order, either all b_i appear before all a_j, or vice-versa.*

In particular, in a collaborative text editor using the algorithm, if two groups of users concurrently insert LtR character sequences at the same position, then in the final list order, the two sequences are not interleaved.

Proof. This follows easily from the tree walk. (A similar result has been formally proven by Kleppmann et al. [6, 5]). □

The rest of the section proves Theorem 2.2. First, we drop the concurrency requirement in our non-interleaving assumption:

Lemma 2.4. *Assuming Definition 2.1, suppose list elements a and $b_1, \ldots, b_k$ satisfy:*

- *a and b_1 have the same left origin.*
- *$b_1, \ldots, b_k$ form a chain of left origins, i.e., the left origin of b_i is b_{i-1} for all $i \geq 2$.*

Then in the final list order, all b_i are on the same side of a, i.e., either $a < b_1, \ldots, b_k$ or $b_1, \ldots, b_k < a$.

Proof. If $a < b_1$, then $a < b_1, \ldots, b_k$, since $b_1 < b_2 < \cdots < b_k$ due to the chain of left origins.

Else $b_1 < a$. It cannot be the case that a is causally greater than b_1, since a's left origin is to the left of b_1 but a is to the right of b_1. Thus a is concurrent to or causally prior to b_1. Since all b_i are causally greater than b_1 (due to the chain of left origins), a is likewise concurrent to or causally prior to all b_i.

More specifically, there must be an index j (possibly 0 or k) such that a is concurrent to $b_1, \ldots, b_j$ and causally prior to $b_{j+1}, \ldots, b_k$. By the non-interleaving assumption, $b_1, \ldots, b_j < a$. Next, when b_{j+1} was inserted, it was aware of both b_j and a, but chose b_j as its left origin instead of a; thus it was inserted to the left of a, i.e., $b_{j+1} < a$. The same holds for the rest of $b_{j+2}, \ldots, b_k$. □

Next, we extend to a tree of b_i's instead of a chain:

Lemma 2.5. *Assuming Definition 2.1, suppose list elements a and $b_1, \ldots, b_k$ satisfy:*

- *a and b_1 have the same left origin.*
- *$b_1, \ldots, b_k$ form a tree of left origins rooted at b_1, i.e., the left origin of b_i is some other b_j for all $i \geq 2$.*

Then in the final list order, all b_i are on the same side of a, i.e., either $a < b_1, \ldots, b_k$ or $b_1, \ldots, b_k < a$.

Proof. For any i, b_i is contained in a chain of the kind described in Lemma 2.4. Thus by that lemma, b_i is on the same side of a as b_1. □

Finally, we prove the theorem.

Proof of Theorem 2.2. To prove that the list order is a tree walk over the left origin tree, it suffices to prove two statements:

(1) Each element is greater than its parent in the tree.

(2) If x and y are siblings in the tree and $x < y$, then the entire subtree rooted at x is less than y.

Statement (1) holds because each element's parent is its left origin, which is lesser by definition. Statement (2) follows from Lemma 2.5 with $x = b_1$ and $y = a$, since siblings in the tree have the same left origin. □

(a) Left origin tree.

(b) Right origin tree.

Figure 1: Left and right origin trees for the example in Theorem 3.1.

3 Doubly Non-Interleaving

By swapping left origins with right in Definition 2.1, we obtain a minimal definition of RtL non-interleaving. It implies RtL analogs of each result in the previous section. In particular, an RtL non-interleaving algorithm's final list order must be a tree walk over the right-origin tree in which each element is ordered *after* its children, for some ordering of siblings in the tree.

Call an algorithm *doubly non-interleaving* if it satisfies both Definition 2.1 and its RtL analog. Such an algorithm must be compatible with tree walks on both the left and right origin trees. Unfortunately, this is impossible in general:

Theorem 3.1. *No list CRDT is doubly non-interleaving.*

Proof. We give a (rather convoluted) counterexample, explained in terms of text editing. Figure 1 shows the final left and right origin trees.

The document starts as a. Concurrently, one user types b after a (yielding ab), while another types c after a (ac). WLOG $b < c$ in the final document order (so merging would yield abc).

After receiving c but not b (state ac), one user types e between a and c (aec), while another types g (agc). Both users then receive b. Note that b and c have the same right origins (the end of the document), $b < c$, and e, g have right origin c; thus by RtL non-interleaving, $b < e, g$. So, the two users see $abec$ and $abgc$.

Next, the user with $abec$ types d between b and e ($abdec$), while concurrently, the user with $abgc$ types f between b and g ($abfgc$).

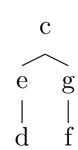
Finally, they merge their changes. Note that b and e have the same left origins (a), $b < e$, and d, f have left origin b; thus by LtR non-interleaving, $d, f < e$. Likewise, $d, f < g$. So, we have

$$a < b < (d, f) < (e, g) < c$$

The allowed final orders are then

$$abdfegc \qquad abdfgec \qquad abfdegc \qquad abfdgec$$

All of these orders interleave de with fg. But this is forbidden by RtL non-interleaving: the right-origin tree contains the subtree



and so the final order on $\{d, e, f, g\}$ must be either $defg$ or $fgde$. □

3.1 Double RGA

The closest we can get to double non-interleaving is to use an algorithm along the lines of:

1. Sort using a tree walk on the left origin tree.

2. For siblings in this tree (which are not sorted by the tree walk), sort using a tree walk on the right origin tree restricted to those siblings (with an arbitrary order on siblings-within-siblings).

This is a novel list CRDT that I call *Double RGA*. The description here is a complete algorithm, but I will elaborate on it more in future work.

Double RGA circumvents the impossibility result because it is not quite RtL non-interleaving: when ordering nodes with different left origins, we don't consider the right origin tree at all, and so RtL non-interleaving can fail for them. In the example of Theorem 3.1, $\{d, e, f, g\}$ don't all have the same left origin, so Double RGA effectively ignores the right origin tree walk for them.

Acknowledgments

I thank Martin Kleppmann for helpful discussions. This work is supported by an NDSEG Fellowship sponsored by the US Office of Naval Research.

References

[1] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. Specification and complexity of collaborative text editing. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, PODC '16, page 259–268, New York, NY, USA, 2016. Association for Computing Machinery.

[2] Basho. Riak datatypes, 2015. `http://github.com/basho`.

[3] Victor Grishchenko. Citrea and swarm: Partially ordered op logs in the browser: Implementing a collaborative editor and an object sync library in javascript. In *Proceedings of the First Workshop on Principles and Practice of Eventual Consistency*, PaPEC '14, New York, NY, USA, 2014. Association for Computing Machinery.

[4] Martin Kleppmann. personal communication.

[5] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. Opsets: Sequential specifications for replicated datatypes. *Archive of Formal Proofs*, May 2018. `https://isa-afp.org/entries/OpSets.html`, Formal proof development.

[6] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. Opsets: Sequential specifications for replicated datatypes (extended version), 2018.

[7] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. Interleaving anomalies in collaborative text editors. In *Proceedings of the 6th Workshop on Principles and Practice of Consistency for Distributed Data*, PaPoC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. Local-first software: You own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, page 154–178, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils. Lseq: An adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, page 37–46, New York, NY, USA, 2013. Association for Computing Machinery.

[10] Nuno Preguiça, Carlos Baquero, and Marc Shapiro. *Conflict-Free Replicated Data Types CRDTs*, pages 1–10. Springer International Publishing, Cham, 2018.

[11] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011.

[12] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. Replicated abstract data types: Building blocks for collaborative applications. *Journal of Parallel and Distributed Computing*, 71(3):354–368, 2011.

[13] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Research Report RR-7506, Inria – Centre Paris-Rocquencourt ; INRIA, January 2011.

[14] Stephane Weiss, Pascal Urso, and Pascal Molli. Logoot: A scalable optimistic replication algorithm for collaborative editing on p2p networks. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 404–412, 2009.