# Interleaving anomalies in collaborative text editors

Martin Kleppmann
mk428@cl.cam.ac.uk
University of Cambridge, UK

Victor B. F. Gomes
vb358@cl.cam.ac.uk
University of Cambridge, UK

Dominic P. Mulligan
Dominic.Mulligan@arm.com
Arm Research, Cambridge, UK

Alastair R. Beresford
arb33@cl.cam.ac.uk
University of Cambridge, UK

## ABSTRACT

Collaborative text editors allow two or more users to concurrently edit a shared document without merge conflicts. Such systems require an algorithm to provide *convergence*, ensuring all clients that have seen the same set of document edits are in the same state. Unfortunately convergence alone does not guarantee that a collaborative text editor is usable. Several published algorithms for collaborative text editing exhibit an undesirable anomaly in which concurrently inserted portions of text with a well-defined order may be randomly interleaved on a character-by-character basis, resulting in an unreadable jumble of letters. Although this anomaly appears to be known informally by some researchers in the field, we are not aware of any published work that fully explains or addresses it. We show that several algorithms suffer from this problem, explain its cause, and also identify a lesser variant of the anomaly that occurs in another algorithm. Moreover, we propose a specification of collaborative text editing that rules out the anomaly, and show how to prevent the lesser anomaly from occurring in one particular algorithm.

## CCS CONCEPTS

• **Theory of computation** → **Distributed algorithms**; • **Software and its engineering** → **Consistency**; • **Human-centered computing** → *Collaborative and social computing systems and tools*; • **Social and professional topics** → *Computer supported cooperative work*; • **Computer systems organization** → *Distributed architectures.*

## KEYWORDS

CRDTs, collaborative text editing, specification, consistency

**ACM Reference Format:**
Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. 2019. Interleaving anomalies in collaborative text editors. In *6th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '19), March 25, 2019, Dresden, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3301419.3323972

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*PaPoC '19, March 25, 2019, Dresden, Germany*
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6276-4/19/03…$15.00
https://doi.org/10.1145/3301419.3323972

## 1 INTRODUCTION

In collaborative software, several users may contribute to a project by creating and editing shared documents such as text documents, spreadsheets, or similar. When a user wishes to view a document, a copy of that document is loaded on the user's computer in a web-browser tab, or in a native app on their device. Any changes made by the user are immediately applied to the local copy of the document on the user's computer, and then asynchronously sent to any other users who have a copy of the document—possibly via a server, which may also store a copy. This collaboration scenario is very similar to the problem of replication in distributed databases: in this context, the shared data is a database rather than a document, and each node that has a copy of the data is called a *replica*.

At a high level, there are two possible ways of managing modifications to documents: either the system enforces that only one user at a time may edit a particular document using a synchronization mechanism (e.g. locks), or the system allows multiple users to edit a document concurrently. This latter case is known as *optimistic replication* [17]. In an optimistic replication setting, several users may make changes at the same time, causing the state of these users' documents to diverge. This is illustrated in Figure 1, where one user changes the text of their document from `'Hello!'` to `'Hello World!'`, while another user concurrently edits the text of their document to read `'Hello! :)'`. In order to ensure that no user input is lost, these concurrent changes must be *merged* into a consistent document—in this example `'Hello World! :)'`.

Merge operations can either be performed manually—the approach used by version control systems such as `git`—or can be automated. Conflict-free Replicated Data Types, or CRDTs [18, 19], have been developed to automate such merges. A CRDT is an abstract datatype whose state can be modified by performing certain operations. For example, a datatype for text editing may represent text as a list of characters, allowing characters to be inserted or deleted anywhere in the document. Local changes are propagated to other replicas, either by encoding the updated state and merging it into remote copies of the document, or by encoding the operations and applying them to remote copies.

### 1.1 Consistency for optimistic replication

CRDTs implement a consistency model called *strong eventual consistency* [6, 19], defined by the following properties:

**Eventual delivery:** An update applied on one correct replica is eventually applied on all correct replicas.

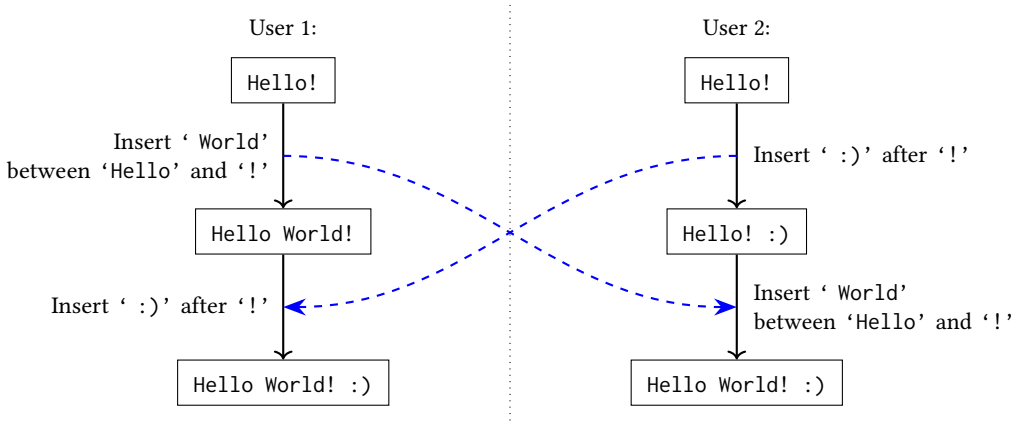


**Figure 1: Simple concurrent text editing example. Solid black lines indicate state changes over time while dashed blue arrows indicate network communication.**

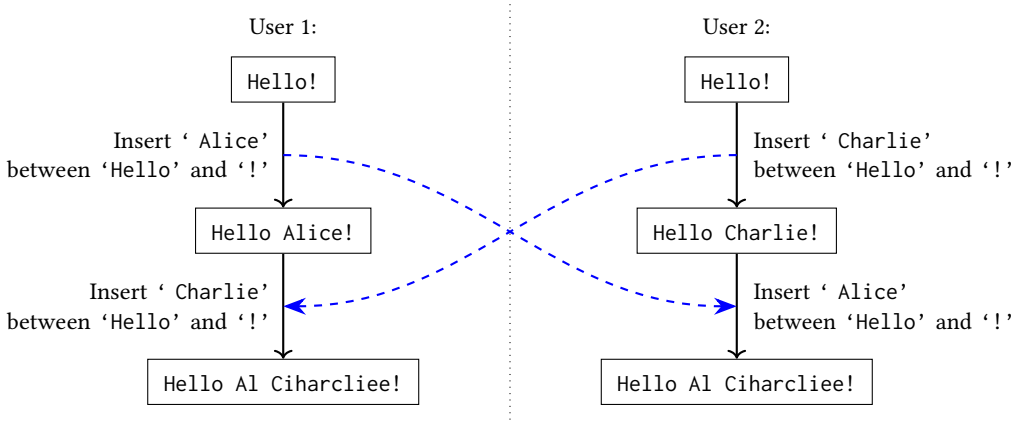


**Figure 2: Two concurrent insertions at the same position are interleaved.**

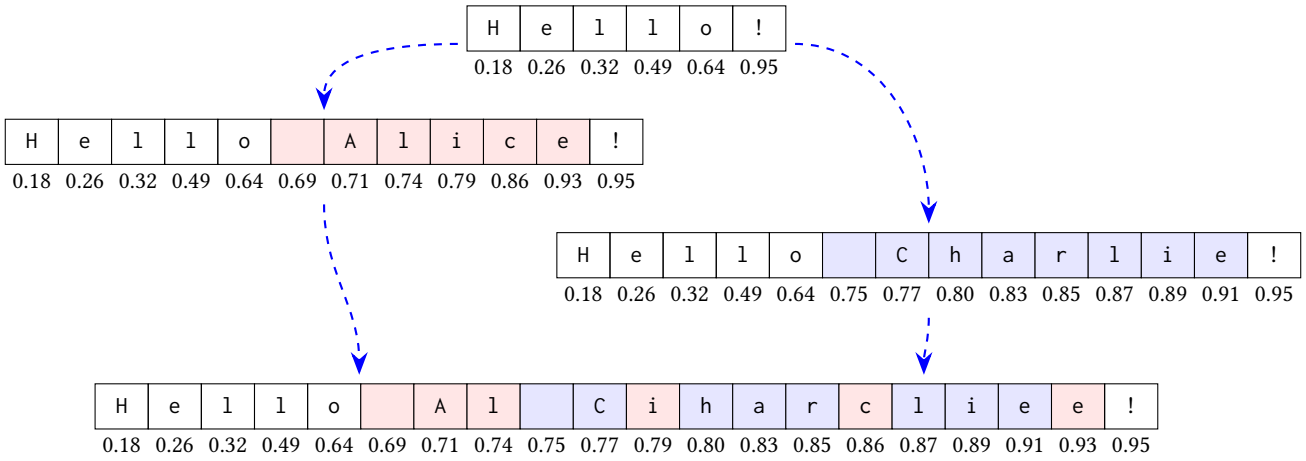


**Figure 3: Interleaving due to character positions taken from a dense identifier set, e.g. the rational numbers $\mathbb{Q}$.**

**Convergence:** If the same set of updates have been applied (possibly in a different order) on two replicas then those two replicas have equivalent state.

**Termination:** All method executions terminate.

In operation-based CRDTs the convergence property is implemented by ensuring that concurrent operations *commute*. For example, consider Figure 1 wherein User 1 first applies the insertion of `' World'` and then applies the insertion of `' :)'` to the document, while User 2 applies the same two insertions in the opposite order. Commutativity of the insertions ensures that the two replicas obtain the same final state.

When replicas mutually merge each others' changes, the convergence property ensures that those replicas obtain the same state. However, we must also specify what that state should be: strong eventual consistency is necessary but it is not sufficient, as it does not capture all of the consistency properties that we require. For example, the convergence property for a document could be met by storing all inserted characters in lexicographical order, but this does not represent a useful document editing system. More is needed in order to ensure the system is useful.

In this paper we examine a particular consistency property for collaborative text editors that has been overlooked in the literature. Notably, failing to satisfy this property may result in arbitrary text interleaving leading to an undesirable (garbled) document state. We highlight existing CRDT algorithms and a specification that exhibit this anomaly, and demonstrate how to fix this flaw.

## 2 THE INTERLEAVING ANOMALY

In this section we discuss an anomaly that can lead to undesirable outcomes when two users concurrently insert text at the same position in a document. For example, in Figure 2, two users are editing a text document that initially reads `'Hello!'`. User 1 changes it to read `'Hello Alice!'`, while concurrently User 2 changes it to `'Hello Charlie!'`. However, when the concurrent edits are merged, the merging algorithm randomly interleaves the two insertions of `' Alice'` and `' Charlie'` character by character, resulting in an unreadable jumble of characters.

Even though this outcome is obviously undesirable, it does sometimes occur in practice. Two published CRDTs for collaborative text editing, Logoot [21, 22] and LSEQ [12, 13], suffer from this problem, as we explain shortly. In prior work [9, 10] we mechanically proved that another text editing CRDT, RGA [16], does not suffer from this problem; however, RGA can exhibit a lesser variant of the anomaly, which we describe in Section 3. We conjecture that Treedoc [15] and WOOT [14] do not suffer from either anomaly, but we leave a rigorous proof of this claim for future work.

The reason why this anomaly occurs with Logoot and LSEQ is illustrated in Figure 3. Conceptually, these algorithms work by assigning every character of the text a unique position identifier from a *dense* ordered set; that is, for any two given identifiers we can find a new, distinct identifier that lies between the two. The order of characters in the text is then given by the order of these identifiers. In Figure 3 we use rational numbers between 0.0 and 1.0 as identifiers. In reality, identifiers in Logoot and LSEQ are paths through a tree, which have the same effect.

We can see in Figure 3 that identifiers are assigned correctly by each of the users: the characters of `' Alice'` and `' Charlie'` are assigned rational numbers between 0.64 and 0.95 (the interval between the preceding `'Hello'` and the following `'!'`) in increasing order. However, the exact values assigned to each character can vary arbitrarily, and since neither user knows about the other user's concurrent insertion, both users spread the identifiers of their insertions across the interval (0.64, 0.95). When merged, the resulting character sequence is an arbitrary interleaving of the two.

We performed tests with open source implementations of Logoot [1, 2] and LSEQ [5, 12], and observed this interleaving anomaly occurring in practice [8]. The problem is even worse if the concurrent insertions are comprised of not just a single word but a paragraph or section. In these cases, interleaving the users' insertions would most likely result in an incomprehensible text that would have to be deleted and rewritten.

Note, however, that this problem does not occur in the example of Figure 1. Here, the merged outcome is unambiguous as the relative ordering of all parts of the text is clear: `' World'` is inserted before the exclamation mark, while the `' :)'` is inserted after it.

### 2.1 Attiya et al.'s specification

In 2016 Attiya et al. [4] proposed $\mathcal{A}_{\mathsf{strong}}$, an implementation-independent specification of collaborative text editing. However, this specification also permits the interleaving anomaly; we therefore argue that it is too weak to be a suitable specification for collaborative text editing. The definition of $\mathcal{A}_{\mathsf{strong}}$ is as follows [4]:

> An abstract execution $A = (H, \mathsf{vis})$ belongs to the *strong list specification* $\mathcal{A}_{\mathsf{strong}}$ if and only if there is a relation $\mathsf{lo} \subseteq \mathsf{elems}(A) \times \mathsf{elems}(A)$, called the *list order*, such that:
>
> (1) Each event $e = do(op, w) \in H$ returns a sequence of elements $w = a_0 \ldots a_{n-1}$, where $a_i \in \mathsf{elems}(A)$, such that
>
> (a) $w$ contains exactly the elements visible to $e$ that have been inserted, but not deleted:
>
> $$\forall a.\ a \in w \iff (do(\mathsf{ins}(a, \_), \_) \leq_{\mathsf{vis}} e) \wedge \neg(do(\mathsf{del}(a), \_) \leq_{\mathsf{vis}} e).$$
>
> (b) The order of the elements is consistent with the list order:
>
> $$\forall i, j.\ (i < j) \implies (a_i, a_j) \in \mathsf{lo}.$$
>
> (c) Elements are inserted at the specified position: if $op = \mathsf{ins}(a, k)$, then $a = a_{\min\{k,\ n-1\}}$.
>
> (2) The list order lo is transitive, irreflexive and total, and thus determines the order of all insert operations in the execution.

The list order relation lo in this definition plays the same role as the position identifiers in Figure 3, and hence this specification permits the same interleaving anomaly. We can correct this flaw in the $\mathcal{A}_{\mathsf{strong}}$ specification and rule out interleaving by introducing an additional clause 1(d):

> (1) Each event $e = do(op, w) \in H$ returns a sequence of elements $w = a_0 \ldots a_{n-1}$, where $a_i \in \mathsf{elems}(A)$, such that
>
> (a) … (c): as before;

(d) Concurrent insertions are not interleaved: that is, for any two sets of insertions $X$ and $Y$

$$X = \{x \mid \exists a.\ x = do(\mathsf{ins}(a, \_), \_) \ \wedge\ x \leq_{\mathsf{vis}} e\}$$
$$Y = \{y \mid \exists a.\ y = do(\mathsf{ins}(a, \_), \_) \ \wedge\ y \leq_{\mathsf{vis}} e\}$$

such that all operations in $X$ and $Y$ are concurrent:

$$\forall x \in X.\ \forall y \in Y.\ \neg(x \leq_{\mathsf{vis}} y) \wedge \neg(y \leq_{\mathsf{vis}} x)$$

if the insertions are at the same location in the document:

$$\exists i, j.\ \{a_k \mid i < k < j\} = \{a \mid do(\mathsf{ins}(a, \_), \_) \in X \cup Y\}$$

then we have either

$$\forall i, j.\ do(\mathsf{ins}(a_i, \_), \_) \in X \ \wedge\ do(\mathsf{ins}(a_j, \_), \_) \in Y \implies i < j \quad \text{or}$$
$$\forall i, j.\ do(\mathsf{ins}(a_i, \_), \_) \in X \ \wedge\ do(\mathsf{ins}(a_j, \_), \_) \in Y \implies j < i.$$

That is, either all $X$ insertions appear before all $Y$ insertions in the document $w = a_0 \ldots a_{n-1}$, or vice versa, but they are never interleaved.

(2) as before.

The additional clause specifically addresses the case of concurrent insertions within the same interval $a_i \ldots a_j$ and rules out interleaving. The existing clause 1(b) ensures that all replicas resolve the insertions to appear in the same order.

## 3 THE LESSER INTERLEAVING ANOMALY

The RGA algorithm for collaborative text editing [16] does not suffer from the anomaly described in Section 2, as we proved in prior work [9, 10]. In particular, if the insertions are made in sequential order (e.g. the string `' Alice'` is inserted by first typing a space, then the letter `'A'`, then the letter `'l'`, etc.), then RGA guarantees that there will be no interleaving. In the scenario of Figure 2, assuming sequential insertions, RGA allows only two possible outcomes of the merge: either `'Hello Alice Charlie!'` or `'Hello Charlie Alice!'` with no mixture of the two permitted.

However, RGA does not satisfy the revised specification of Section 2.1 because it allows a lesser anomaly: text may be interleaved if insertions are not sequential. This lesser anomaly is illustrated in Figure 4. In this example, User 1 first positions the cursor between `'Hello'` and the exclamation mark, types the word `' reader'`, then *moves the cursor back* to a position immediately after `'Hello'`, and types the word `' dear'`.

In RGA an insertion is anchored to the existing character that immediately precedes the character to be inserted. Thus, in Figure 4, the first character of `' reader'` and the first character of `' dear'` are both anchored to the last character of `'Hello'`. When User 2 makes a concurrent insertion of `' Alice'`, it is also anchored to the last character of `'Hello'`. This tree of anchoring relationships is illustrated in Figure 5 (Attiya et al. call this structure a *timestamped insertion tree* [4], while Grishchenko calls it a *causal tree* [7]).

When multiple insertions are anchored to the same character, they are sorted in descending timestamp order. In the example, we can assume that the timestamp of the insertion of `' dear'` is greater than that of `' reader'`, since these insertions were performed sequentially by the same user. However, we do not know the timestamp of `' Alice'` relative to the other two insertions, and hence that insertion can be ordered arbitrarily relative to other insertions with the same anchor. Thus, in this example, RGA allows three possible outcomes of the merge:

(1) `'Hello dear reader Alice!'`
(2) `'Hello dear Alice reader!'`
(3) `'Hello Alice dear reader!'`

Although RGA rules out random character-by-character interleaving, it does still allow the word `'Alice'` to be interleaved between the two insertion sequences by User 1. We characterise this as a lesser form of the anomaly described in Section 2.

The worst case for RGA occurs if the user types all characters in the reverse order of their appearance in the document, i.e. the document is typed back to front. In this case, all characters would be anchored to the head of the document, ordered only by timestamp, and thus arbitrary character-level interleaving could occur. Whilst this editing pattern is unlikely to occur in realistic editing scenarios, it is a case that must be considered by formal consistency models for collaborative text editors.

### 3.1 Fixing interleaving in RGA

RGA assigns a unique logical timestamp to every operation, and treats that timestamp as the identifier for that operation. Roh et al.'s original definition of RGA used a custom S4Vector datatype as timestamp [16], and subsequent presentations of the algorithm [18] have used Lamport timestamps [11] instead. Our RGA fix works by changing the algorithm to use a new timestamp definition.

In Attiya et al.'s formulation of RGA [4], an insertion operation is represented by a triple $(a, t, r)$ where $a$ is the character being inserted, $t$ is the timestamp of the operation, and $r$ is the timestamp of the *reference character* (the timestamp of the operation that inserted the immediate predecessor character at the time the insertion was performed, or the distinguished value *head* if the insertion was performed at the beginning of the document). Deletion operations are performed by marking a character as deleted but retaining its position in the document as a *tombstone*. Since deletions do not affect identifier order in the document, we ignore them henceforth.

We extend this definition to represent each insertion operation by a 4-tuple $(a, t, r, e)$ where $a$, $t$ and $r$ are defined as before, and $e$ is the set of timestamps of all insertion operations with the same reference character $r$ at the time the insertion was performed (not including $t$ itself). Let $I$ be the set of insertion operations that have been performed on a document at a given point in time. To insert a character $a$ after reference character $r$ in this document, we create a new timestamp $t$, create an insertion operation, and add it to $I$:

$$I' = I \ \cup\ \left\{ (a,\ t,\ r,\ \{t' \mid \exists a', e'.\ (a', t', r, e') \in I\}\ ) \right\}$$

In the example of Figure 5, if the timestamp of the last character of `'Hello'` is $\tau_0$, the set of insertions contains $(\text{`!'}, \tau_1, \tau_0, \{\})$ for the exclamation mark, $(\text{`\ '}, \tau_2, \tau_0, \{\tau_1\})$ for the first character of `' reader'`, $(\text{`\ '}, \tau_3, \tau_0, \{\tau_1\})$ for the first character of `' Alice'`, and $(\text{`\ '}, \tau_4, \tau_0, \{\tau_1, \tau_2\})$ for the first character of `' dear'`.

In order to use this insertion 4-tuple in RGA we need to define a total order that is used to sort insertions with the same reference character. Consider two distinct insertion operations $ins_1$ and $ins_2$

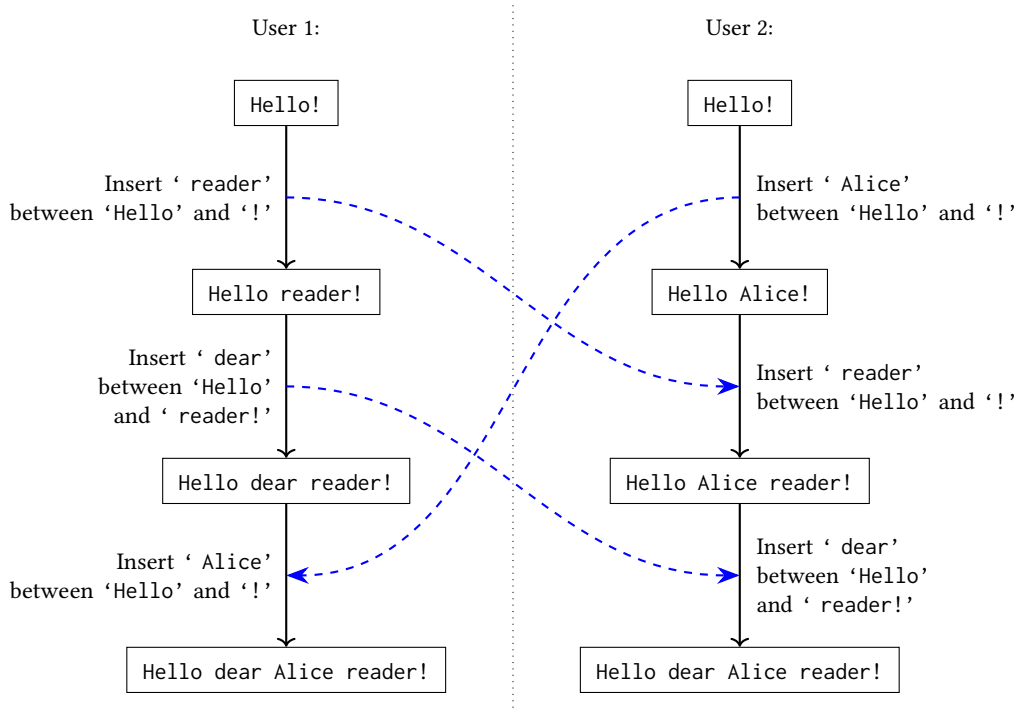


**Figure 4: The lesser interleaving anomaly that can occur with RGA.**

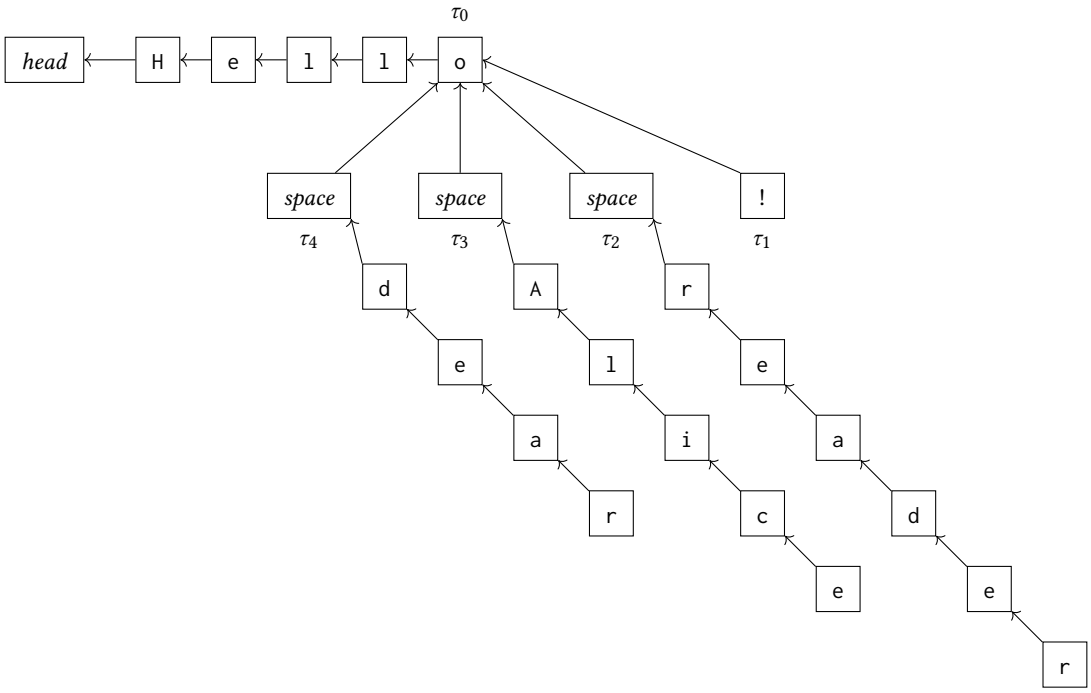


**Figure 5: The tree structure underlying the RGA example in Figure 4: each node represents a character, and its parent is the immediate predecessor character at the time it was inserted. The document state corresponds to a depth-first pre-order traversal over this tree, with sibling nodes visited in descending timestamp order ($\tau_4 > \tau_3 > \tau_2 > \tau_1$).**

with the same reference character $r$:

$$ins_1 = (a_1, t_1, r, e_1) \quad \text{and} \quad ins_2 = (a_2, t_2, r, e_2).$$

First observe that if $ins_1$ happened before $ins_2$ we must have $t_1 \in e_2$, and vice versa. In this case we make the ordering of operations consistent with the *happens-before* relation:

$$(a_1, t_1, r, e_1) < (a_2, t_2, r, e_2) \quad \text{if } t_1 \in e_2$$
$$(a_2, t_2, r, e_2) < (a_1, t_1, r, e_1) \quad \text{if } t_2 \in e_1$$

Otherwise $ins_1$ and $ins_2$ are concurrent, and we have $t_1 \notin e_2$ and $t_2 \notin e_1$. From this we can deduce that

$$\{t_1\} \cup e_1 - e_2 \neq \{\} \quad \text{and} \quad \{t_2\} \cup e_2 - e_1 \neq \{\}.$$

As these sets are nonempty and we have a total ordering on timestamps, each of them has a unique minimal element:

$$m_1 = \min(\{t_1\} \cup e_1 - e_2) \quad \text{and} \quad m_2 = \min(\{t_2\} \cup e_2 - e_1)$$

The timestamps $m_1$ and $m_2$ identify the first operations at which the editing histories of $ins_1$ and $ins_2$ diverged. From the definition above it follows that $m_1 \neq m_2$. We can now define the ordering of concurrent operations based on the relative ordering of $m_1$ and $m_2$:

$$(a_1, t_1, r, e_1) < (a_2, t_2, r, e_2) \quad \text{if } m_1 < m_2$$
$$(a_2, t_2, r, e_2) < (a_1, t_1, r, e_1) \quad \text{if } m_2 < m_1$$

This order has the property that all of the operations in a particular editing session are grouped, so that they are either all less than or all greater than the operations in a different, concurrent editing session. Hence, all characters inserted during a particular editing session are grouped together in the final document, and interleaving of insertions from concurrent editing sessions is prevented. We conjecture that applying this construction to RGA results in a CRDT without interleaving, as required by our specification in Section 2.1. A formal proof of this conjecture is left for future work.

Our construction increases the memory and network bandwidth used by the CRDT, due to the additional timestamp set $e$ that needs to be included in each operation. However, the sets are expected to be small, since there are normally not many operations with the same reference character, regardless of the length of the document.

## 4 CONCLUSIONS

Several published CRDT algorithms for collaborative text editing exhibit an undesirable anomaly in which concurrently inserted portions of text may be randomly interleaved on a character-by-character basis, resulting in an unreadable jumble of letters. We highlighted two cases of undesirable interleaving, one affecting Logoot and LSEQ, and the other affecting RGA.

The interleaving anomaly in Logoot and LSEQ has been independently pointed out in our draft manuscript [9], by Sun et al. [20], and by a Stack Overflow user [3]. From conversations with various members of the CRDT community it appears that the anomaly has been known in the community folklore for some time, but to our knowledge there is no published work that clearly explains the problem or proposes solutions.

In this paper we proposed an improved specification of collaborative text editing that rules out these interleaving anomalies, and proposed a modification to the RGA algorithm that, we conjecture, rules out the interleaving anomaly in this algorithm.

## ACKNOWLEDGMENTS

This work was supported by The Boeing Company and the EPSRC "REMS: Rigorous Engineering for Mainstream Systems" programme grant (EP/K008528).

## REFERENCES

[1] Mehdi Ahmed-Nacer, Claudia-Lavinia Ignat, Gérald Oster, Hyun-Gul Roh, and Pascal Urso. 2011. Evaluating CRDTs for real-time document editing. In *11th ACM Symposium on Document Engineering (DocEng)*. 103–112. https://doi.org/10.1145/2034691.2034717

[2] Mehdi Ahmed-Nacer, Gérald Oster, and Pascal Urso. [n. d.]. Java benchmarker of optimistic replication algorithms. https://github.com/PascalUrso/ReplicationBenchmark

[3] Archagon. 2017. Logoot CRDT: interleaving of data on concurrent edits to the same spot? https://stackoverflow.com/questions/45722742/logoot-crdt-interleaving-of-data-on-concurrent-edits-to-the-same-spot

[4] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. 2016. Specification and Complexity of Collaborative Text Editing. In *ACM Symposium on Principles of Distributed Computing (PODC)*. 259–268. https://doi.org/10.1145/2933057.2933090

[5] Chat-Wane. [n. d.]. LSEQTree. https://github.com/Chat-Wane/LSEQTree

[6] Victor B F Gomes, Martin Kleppmann, Dominic P Mulligan, and Alastair R Beresford. 2017. Verifying strong eventual consistency in distributed systems. *Proceedings of the ACM on Programming Languages (PACMPL)* 1, OOPSLA (Oct. 2017). https://doi.org/10.1145/3133933

[7] Victor Grishchenko. 2014. Citrea and Swarm: Partially ordered op logs in the browser. In *1st Workshop on Principles and Practice of Eventual Consistency (PaPEC)*. https://doi.org/10.1145/2596631.2596641

[8] Martin Kleppmann. 2019. Research data supporting "Interleaving anomalies in collaborative text editors" [Dataset]. https://doi.org/10.17863/CAM.37617

[9] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. 2018. OpSets: Sequential Specifications for Replicated Datatypes (Extended Version). https://arxiv.org/abs/1805.04263

[10] Martin Kleppmann, Victor B. F. Gomes, Dominic P. Mulligan, and Alastair R. Beresford. 2018. OpSets: Sequential Specifications for Replicated Datatypes (Proof Document). https://www.isa-afp.org/entries/OpSets.html

[11] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 7 (July 1978), 558–565. https://doi.org/10.1145/359545.359563

[12] Brice Nédelec, Pascal Molli, and Achour Mostefaoui. 2016. CRATE: Writing Stories Together with our Browsers. In *25th International World Wide Web Conference (WWW)*. 231–234. https://doi.org/10.1145/2872518.2890539

[13] Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils. 2013. LSEQ: an Adaptive Structure for Sequences in Distributed Collaborative Editing. In *13th ACM Symposium on Document Engineering (DocEng)*. 37–46. https://doi.org/10.1145/2494266.2494278

[14] Gérald Oster, Pascal Urso, Pascal Molli, and Abdessamad Imine. 2006. Data Consistency for P2P Collaborative Editing. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*. https://doi.org/10.1145/1180875.1180916

[15] Nuno Preguiça, Joan Manuel Marquès, Marc Shapiro, and Mihai Letia. 2009. A commutative replicated data type for cooperative editing. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS)*. https://doi.org/10.1109/ICDCS.2009.20

[16] Hyun-Gul Roh, Myeongjae Jeon, Jin-Soo Kim, and Joonwon Lee. 2011. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel and Distrib. Comput.* 71, 3 (2011), 354–368. https://doi.org/10.1016/j.jpdc.2010.12.006

[17] Yasushi Saito and Marc Shapiro. 2005. Optimistic Replication. *Comput. Surveys* 37, 1 (March 2005), 42–81. https://doi.org/10.1145/1057977.1057980

[18] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. *A comprehensive study of Convergent and Commutative Replicated Data Types.* Technical Report 7506. INRIA. http://hal.inria.fr/inria-00555588/

[19] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-free Replicated Data Types. In *13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS)*. 386–400. https://doi.org/10.1007/978-3-642-24550-3_29

[20] Chengzheng Sun, David Sun, Agustina, and Weiwei Cai. 2018. Real Differences between OT and CRDT for Co-Editors. https://arxiv.org/abs/1810.02137

[21] Stéphane Weiss, Pascal Urso, and Pascal Molli. 2009. Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS)*. 404–412. https://doi.org/10.1109/ICDCS.2009.75

[22] Stéphane Weiss, Pascal Urso, and Pascal Molli. 2010. Logoot-Undo: Distributed Collaborative Editing System on P2P networks. *IEEE Transactions on Parallel and Distributed Systems* 21, 8 (Jan. 2010), 1162–1174. https://doi.org/10.1109/TPDS.2009.173