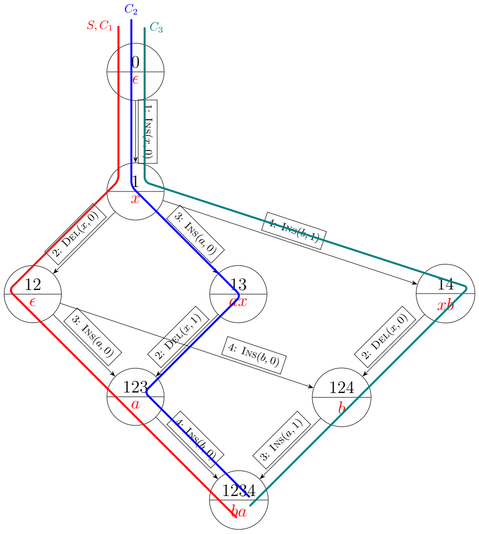

Figure 2: The same final *n*-ary ordered state space constructed by CJupiter for each replica under the schedule of Figure 1. Each replica behavior corresponds to a path going through this state space.

Proposition 3. *In* CJupiter*, the replicas that have processed the same set of operations have the same n-ary ordered state space.*

Jupiter is similar to CJupiter with three major differences: *1)* For a client/server system with n clients, Jupiter [8] maintains $2n$ ***2D state spaces***, each consisting of a *local* dimension and a *global* dimension. In particular, the server maintains n 2D state spaces, one for each client; *2)* In xForm(op : $Op, d \in \{\textit{LOCAL}, \textit{GLOBAL}\}$) of Jupiter, the operation sequence with which op transforms is determined by the parameter d; *3)* In Jupiter, the server propagates the *transformed* operations to other clients.

4 CJUPITER IS EQUIVALENT TO JUPITER

We prove that CJupiter is equivalent to Jupiter from perspectives of both the server and clients. ***At the server***, the n-ary ordered state space CSS_s of CJupiter equals the *union* (in terms of graphs as sets of vertices and edges) of all 2D state spaces maintained at the server for each client in Jupiter. The equivalence of ***clients*** follows since the final transformed operations executed at each client in Jupiter and CJupiter are the same (although the original operations are propagated in CJupiter). Thus, we have that

Theorem 4. *Under the same schedule, the behaviors of corresponding replicas in* CJupiter *and* Jupiter *are the same.*

5 CJUPITER SATISFIES THE WEAK LIST SPECIFICATION

The following theorem, together with Theorem 4, solves the conjecture of Attiya et al. [2].

Theorem 5. CJupiter *satisfies the weak list specification $\mathcal{A}_{weak}$.*

Proof. For each execution α of CJupiter, we first construct an abstract execution $A = (H, \text{vis})$ with vis $= \xrightarrow{\text{hb}_\alpha}$. It is easy to prove the conditions 1(a) and 1(c) of $\mathcal{A}_{\text{weak}}$. Then, we define the ***list order relation*** lo: For $a, b \in \text{elems}(A)$, $a \xrightarrow{\text{lo}} b$ iff there exists an event $e \in \alpha$ with returned list w such that a precedes b in w. By definition, lo satisfies conditions 1(b).

It remains to show the *irreflexivity* of lo, which is equivalent to the ***pairwise state compatibility property***: lo is irreflexive iff any two list states w_1 and w_2 in A are compatible, namely, for any two common elements a and b of w_1 and w_2, their relative orderings are the same in w_1 and w_2. By Proposition 3, it suffices to show that the state space CSS_s at the server satisfies the pairwise state compatibility property. Given a pair of states/vertices in CSS_s, we consider the paths to them from their LCA. [2]

Lemma 6. *Every pair of vertices in CSS_s has a unique LCA.*

Lemma 7. *Let v_0 be the unique LCA of a pair of vertices v_1 and v_2 in CSS_s. Then, the path $P_{v_0 \rightsquigarrow v_1}$ from v_0 to v_1, as well as $P_{v_0 \rightsquigarrow v_2}$ from v_0 to v_2, is* ***simple****, namely, there are no duplicate operations along it. Furthermore, the set of operations $O_{v_0 \rightsquigarrow v_1}$ along $P_{v_0 \rightsquigarrow v_1}$ is* ***disjoint*** *from the set of operations $O_{v_0 \rightsquigarrow v_2}$ along $P_{v_0 \rightsquigarrow v_2}$.*

The desired pairwise state compatibility property follows, when we take the common vertex v_0 in the next Lemma as the LCA of the two vertices v_1 and v_2 under consideration.

Lemma 8. *Let $P_{v_0 \rightsquigarrow v_1}$ and $P_{v_0 \rightsquigarrow v_2}$ be two paths from vertex v_0 to vertices v_1 and v_2, respectively in CSS_s. If they are* ***disjoint simple paths****, then the list states of v_1 and v_2 are* ***compatible****.*

□

6 ACKNOWLEDGMENTS

This work is supported by the National 973 Program of China (2015CB352202), the National Natural Science Foundation of China (61702253), and the Fundamental Research Funds for the Central Universities (020214380031).

REFERENCES

[1] [n. d.]. Apache Wave. https://incubator.apache.org/wave/.

[2] Hagit Attiya, Sebastian Burckhardt, Alexey Gotsman, Adam Morrison, Hongseok Yang, and Marek Zawirski. 2016. Specification and complexity of collaborative text editing. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing (PODC '16)*. ACM, 259–268.

[3] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. 2014. Replicated Data Types: Specification, Verification, Optimality. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, 271–284.

[4] The Apache Wave Foundation. 2015. *Apache Wave (incubating) Protocol Documentation (Release 0.4)*.

[5] Google. 2010. What's different about the new Google Docs: Making collaboration fast. https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs.html.

[6] Leslie Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM* 21, 7 (July 1978), 558–565.

[7] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. 1995. High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST '95)*. ACM, 111–120.

[8] Yi Xu, Chengzheng Sun, and Mo Li. 2014. Achieving Convergence in Operational Transformation: Conditions, Mechanisms and Systems. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '14)*. ACM, 505–518.

[2] The LCA (Lowest Common Ancestor) of two vertices v_1 and v_2 in CSS_s, which is a DAG, is the lowest (i.e., deepest) vertex that has both v_1 and v_2 as descendants.