# Certified Mergeable Replicated Data Types

Kartik Nagar

Joint work with Vimala S, Adharsh Kamath and KC Sivaramakrishnan.

*To appear in* PLDI 22.

# Outline

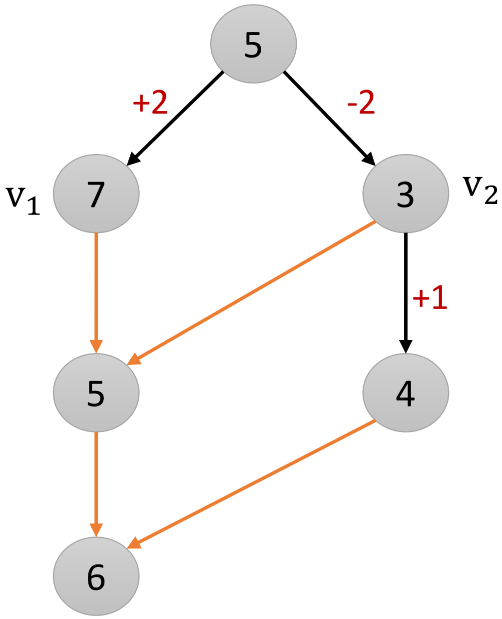
- Introduction
  - Replicated Systems
  - MRDT
- MRDT Verification Problem
- Our proposed technique
- Example
- Experimental Results
- Conclusion

# Replicated Systems

- Creating multiple replicas of data, independently operated, potentially geo-distributed.
- Many benefits
  - Fault Tolerance
  - Availability
  - Low latency for geo-distributed clients
- How to safely write applications for replicated systems?
  - Programming would be easier if it appears as a single, ‘centralized’ system.
  - Unfortunately, this incurs massive synchronization cost.
- Instead, we have a library of basic replicated data types with slightly ‘weaker’ semantics.

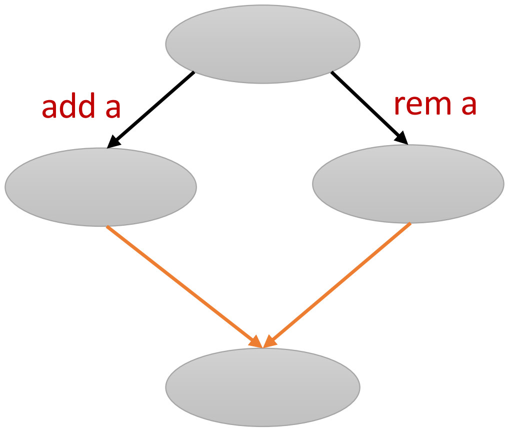
# Mergeable Replicated Data Types (MRDTs)

- Version-control model of replication.
- Three-way merge function – 2 concurrent versions and their Lowest Common Ancestor (LCA) between them.
- Consider the counter MRDT:




$$\text{merge lca } v_1 v_2 = \text{lca} + (v_1 - \text{lca}) + (v_2 - \text{lca})$$

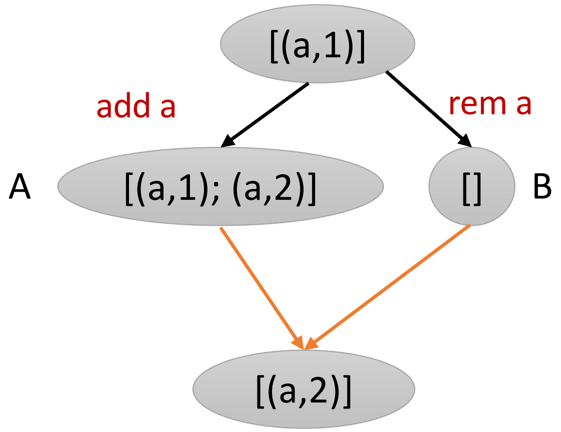
# Set RDT




rd should return a

Desired specification: Add wins

# Observed-Remove Set MRDT




## Implementation

$$D_\tau = (\Sigma, \sigma_0, do, merge)$$

1: $\Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N})$

2: $\sigma_0 = \{\}$

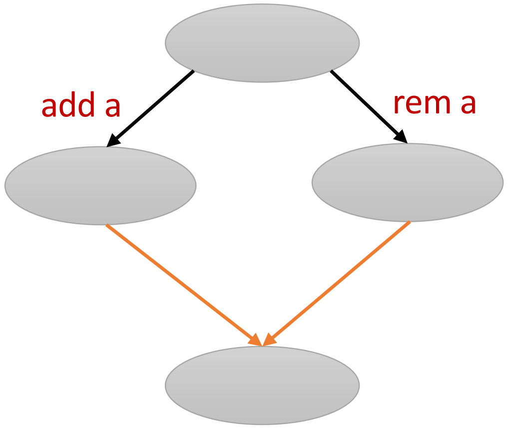
3: $do(rd, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\})$

4: $do(add(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot)$

5: $do(remove(a), \sigma, t) = (\{e \in \sigma \mid fst(e) \neq a\}, \bot)$

6: $merge(\sigma_{lca}, \sigma_a, \sigma_b) =$
$(\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca})$

# Observed-Remove Set Specification



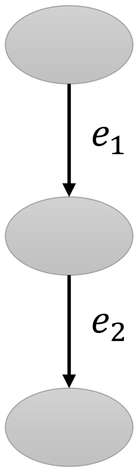

rd should return a

Abstract state

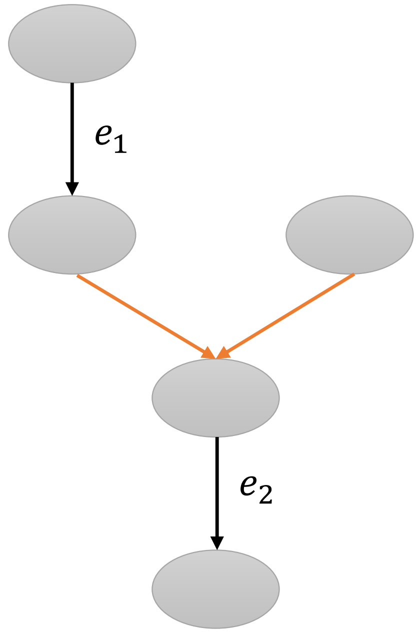
$I = \langle E, oper, rval, time, vis \rangle$

- $E$ is the set of events
- $oper: E \rightarrow Op$
- $rval: E \rightarrow Val$
- $time: E \rightarrow \mathbb{N}$
- $vis \subseteq E \times E$

# Visibility Relation in Abstract State

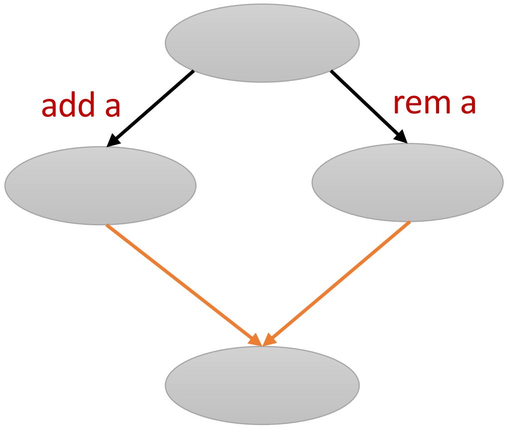



$e_1 \xrightarrow{vis} e_2$




$e_1 \xrightarrow{vis} e_2$

# Observed-Remove Set Specification




rd should return a

Abstract state

$I = \langle E, oper, rval, time, vis \rangle$

- $E$ is the set of events
- $oper: E \to Op$
- $rval: E \to Val$
- $time: E \to \mathbb{N}$
- $vis \subseteq E \times E$

Specification

$$\mathcal{F}_{orset}(\mathrm{rd}, \langle E, oper, rval, time, vis \rangle) = \{a \mid \exists e \in E.\ oper(e)$$

$$= \mathrm{add}(a) \wedge \neg(\exists f \in E.\ oper(f) = \mathrm{remove}(a) \wedge e \xrightarrow{vis} f)\}$$

# The problem

Implementation

$$\Sigma = \mathcal{P}(\mathbb{N} \times \mathbb{N})$$
$$\sigma_0 = \{\}$$
$$do(rd, \sigma, t) = (\sigma, \{a \mid (a, t) \in \sigma\})$$
$$do(add(a), \sigma, t) = (\sigma \cup \{(a, t)\}, \bot)$$
$$do(remove(a), \sigma, t) = (\{e \in \sigma \mid fst(e) \neq a\}, \bot)$$
$$merge(\sigma_{lca}, \sigma_a, \sigma_b) =$$
$$(\sigma_{lca} \cap \sigma_a \cap \sigma_b) \cup (\sigma_a - \sigma_{lca}) \cup (\sigma_b - \sigma_{lca})$$

Specification

$$\mathcal{F}_{orset}(\mathrm{rd}, \langle E, oper, rval, time, vis \rangle) = \{a \mid \exists e \in E.\ oper(e)$$
$$= \mathrm{add}(a) \wedge \neg(\exists f \in E.\ oper(f) = \mathrm{remove}(a) \wedge e \xrightarrow{vis} f)\}$$

1. Does the implementation satisfy the specification?
2. Does the implementation ensure convergence?
   - Two replicas which have witnessed the same set of events must have the same state.

# Our Contributions

- We propose a simulation-based verification procedure for showing functional correctness and convergence for MRDTs.
- We mechanize and automate the complete verification process using F*.
- We propose a new, weaker notion of convergence modulo observable behavior which permits more efficient MRDT implementations.
- We have built a library of efficient and verified MRDTs for common data structures such as set, map, queue, flag, etc.

# Replication-aware Simulation Relation[1]

- $\mathcal{R}_{sim}(I, \sigma)$ relates an abstract state $I$ with concrete state $\sigma$.
  - $\mathcal{R}_{sim}$ is the glue relating the concrete and abstract states, as well as the implementation and specification
- Verification using $\mathcal{R}_{sim}$ is done in two steps:
  1. We show that $\mathcal{R}_{sim}$ holds in all executions in an inductive fashion.
  2. We show that $\mathcal{R}_{sim}$ is sufficient to discharge the specification and convergence requirements.

1. *Burckhardt et. al.* Replicated Data Types: Specification, Verification and Optimality. POPL 2014*.*

# OR-Set MRDT Simulation Relation

$$\mathcal{R}_{sim}(I, \sigma) \iff (\forall a, t.\ (a, t) \in \sigma \iff$$
$$(\exists e \in I.E \wedge I.\ oper(e) = add(a) \wedge I.time(e) = t \wedge$$
$$\neg(\exists f \in I.E \wedge I.\ oper(f) = remove(a) \wedge e \xrightarrow{vis} f)))$$

# Verification using $\mathcal{R}_{sim}$: Step-1

We show that $\mathcal{R}_{sim}$ holds inductively at every step in every execution

1. Verifying operations

$$
\begin{tikzcd}
I \arrow[r, "do^{\#}"] \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"'] & I' \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] \\
\sigma \arrow[r, "\mathcal{D}_\tau.do"'] & \sigma'
\end{tikzcd}
$$

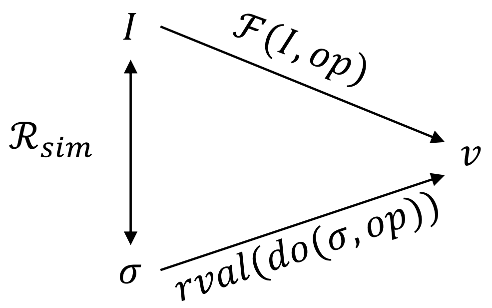
2. Verifying merge

$$
\begin{tikzcd}
\sigma_a \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"'] \arrow[ddr] & I_{lca} \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] & \sigma_b \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] \arrow[ddl] \\
I_a \arrow[dddr] & \sigma_{lca} \arrow[d] & I_b \arrow[dddl] \\
 & \mathcal{D}_\tau.merge(\sigma_{lca}, \sigma_a, \sigma_b) \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] & \\
 & merge^{\#}(I_a, I_b) &
\end{tikzcd}
$$

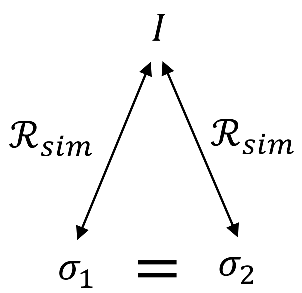
# Verification using $\mathcal{R}_{sim}$: Step-2

We show that $\mathcal{R}_{sim}$ is sufficient to prove specification and convergence

3. Verifying specification




4. Verifying convergence



# Store Properties

$\Psi_{ts}$ asserts increasing timestamps according to the visibility relation

| | |
|---|---|
| $\Psi_{ts}(I)$ | $\forall e, e' \in I.E.\ e \xrightarrow{I.vis} e' \Rightarrow I.time(e) < I.time(e')$ <br> $\land \forall e, e' \in I.E.\ I.time(e) = I.time(e') \Rightarrow e = e'$ |
| $\Psi_{lca}(I_l, I_a, I_b)$ | $I_l.E = I_a.E \cap I_b.E$ <br> $\land\ I_l.vis = I_a.vis_{\mid I_l.E} = I_b.vis_{\mid I_l.E}$ |

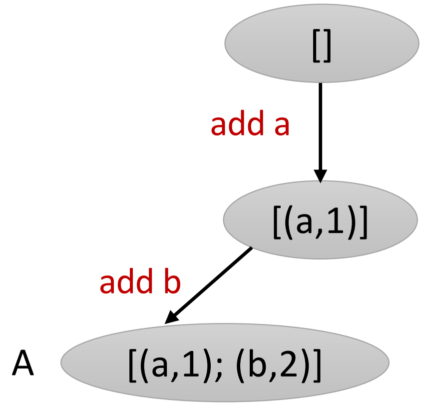
$\Psi_{lca}$ asserts that events in LCA are present in both the branches, with the same visibility relation

We assume the store properties while proving $\mathcal{R}\_sim$

# Example: Verifying $\mathcal{R}_{sim}$ for OR-Set MRDT

Simulation Relation:

$$\mathcal{R}_{sim}(I,\sigma) \iff (\forall a,t.\ (a,t) \in \sigma \iff$$
$$(\exists e \in I.E \wedge I.\,oper(e) = add(a) \wedge I.time(e) = t \wedge$$
$$\neg(\exists f \in I.E \wedge I.\,oper(f) = remove(a) \wedge e \xrightarrow{vis} f)))$$



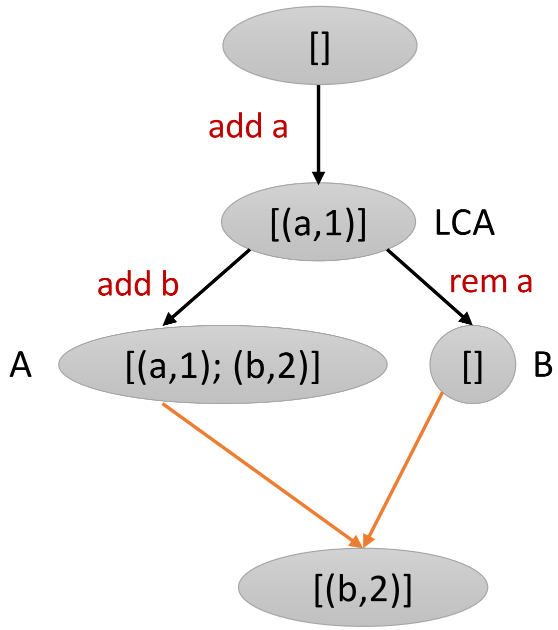

$$\begin{tikzcd} I \arrow[r, "do^{\#}"] \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"'] & I' \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] \\ \sigma \arrow[r, "\mathcal{D}_{\tau}.do"'] & \sigma' \end{tikzcd}$$

# Example: Verifying $\mathcal{R}_{sim}$ for OR-Set MRDT

Simulation Relation:

$$\mathcal{R}_{sim}(I,\sigma) \iff (\forall a,t.\ (a,t) \in \sigma \iff$$
$$(\exists e \in I.E \land I.\,oper(e) = add(a) \land I.time(e) = t \land$$
$$\neg(\exists f \in I.E \land I.\,oper(f) = remove(a) \land e \xrightarrow{vis} f)))$$

$$\begin{tikzcd}
\sigma_a \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"'] \arrow[ddr] & I_{lca} \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] & \sigma_b \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] \arrow[ddl] \\
I_a \arrow[ddr] & \sigma_{lca} \arrow[d] & I_b \arrow[ddl] \\
 & \mathcal{D}_\tau.merge(\sigma_{lca},\sigma_a,\sigma_b) \arrow[d, leftrightarrow, "\mathcal{R}_{sim}"] & \\
 & merge^{\#}(I_a, I_b) &
\end{tikzcd}$$



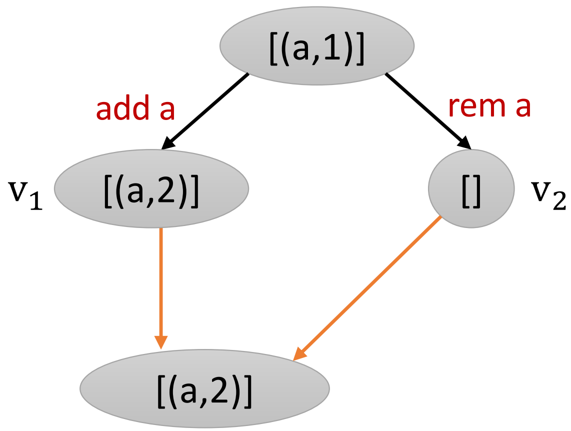

Store Property:

| | |
|---|---|
| $\Psi_{lca}(I_l, I_a, I_b)$ | $I_l.E = I_a.E \cap I_b.E$ $\land\ I_l.vis = I_a.vis_{\mid I_l.E} = I_b.vis_{\mid I_l.E}$ |

# Efficient OR-Set implementations

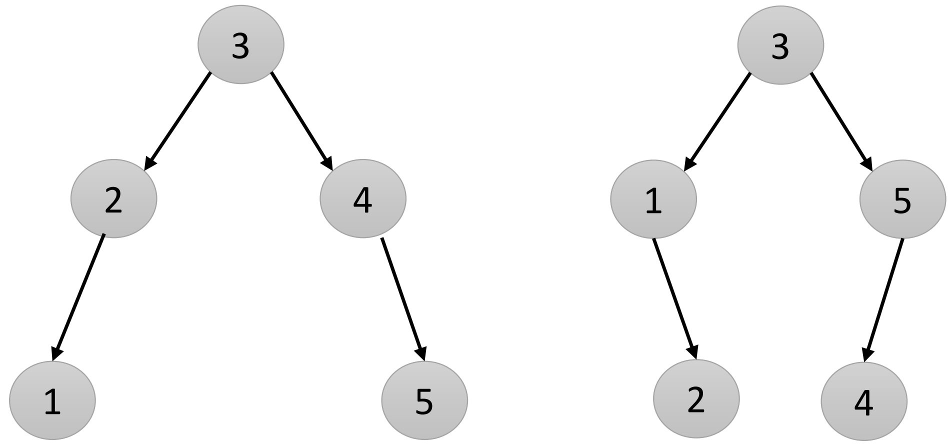
**Space-efficient version**

- Keeps a single version of an element
- Otherwise, it is the same as the original OR-Set MRDT.




**Space & time-efficient version**

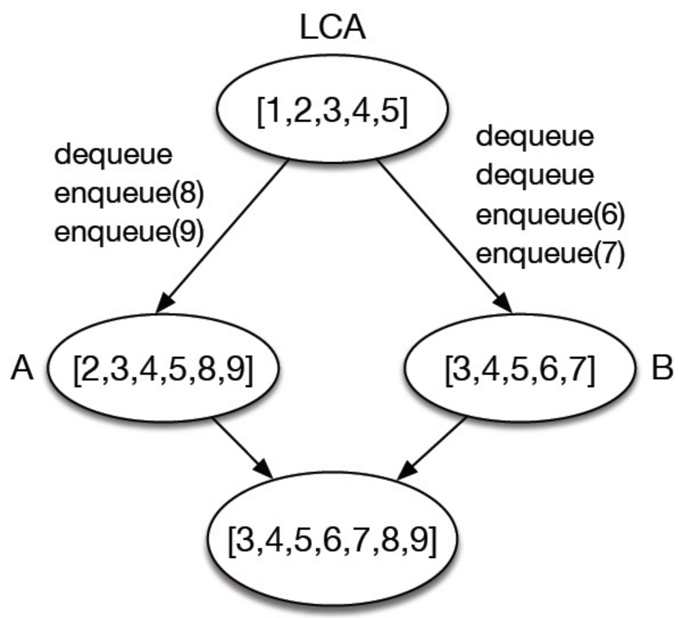
- Stores the set internally as a Binary Search Tree instead of a list
- Much better performance for $rd$ queries.
- We can only guarantee convergence modulo observable behavior.



# Peepul: Library of Verified MRDTs in F*

| MRDTs verified | #Lines code | #Lines proof | #Lemmas | Verif. time (s) |
|---|---|---|---|---|
| Increment-only counter | 6 | 43 | 2 | 3.494 |
| PN counter | 8 | 43 | 2 | 23.211 |
| Enable-wins flag | 20 | 58 | 3 | 1074 |
| | | 81 | 6 | 171 |
| | | 89 | 7 | 104 |
| LWW register | 5 | 44 | 1 | 4.21 |
| G-set | 10 | 23 | 0 | 4.71 |
| | | 28 | 1 | 2.462 |
| | | 33 | 2 | 1.993 |
| G-map | 48 | 26 | 0 | 26.089 |
| Mergeable log | 39 | 95 | 2 | 36.562 |
| OR-set (§2.1.1) | 30 | 36 | 0 | 43.85 |
| | | 41 | 1 | 21.656 |
| | | 46 | 2 | 8.829 |
| OR-set-space (§2.1.2) | 59 | 108 | 7 | 1716 |
| OR-set-spacetime | 97 | 266 | 7 | 1854 |
| Queue | 32 | 1123 | 75 | 4753 |

# Verified Queue MRDT



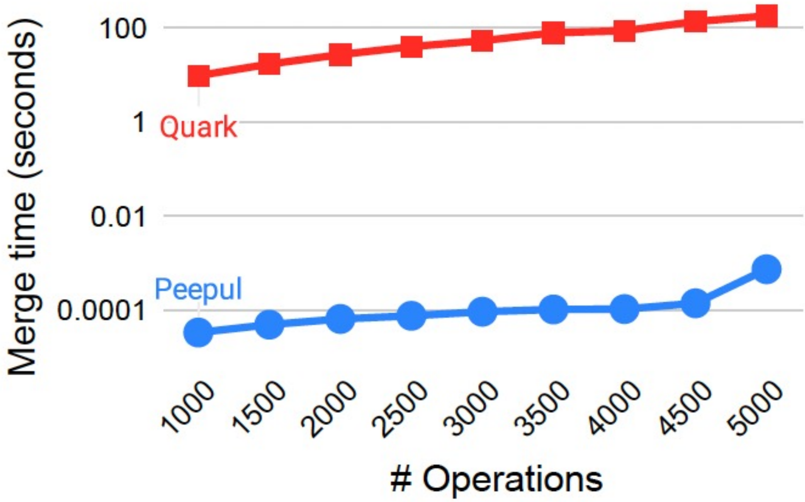

At-least-once dequeue semantics

# Specification of the Queue MRDT

$$\mathrm{match}_I(e_1, e_2) \Leftrightarrow I.oper(e_1) = enqueue(a)$$
$$\wedge\, I.oper(e_2) = dequeue \wedge a = I.rval(e_2)$$

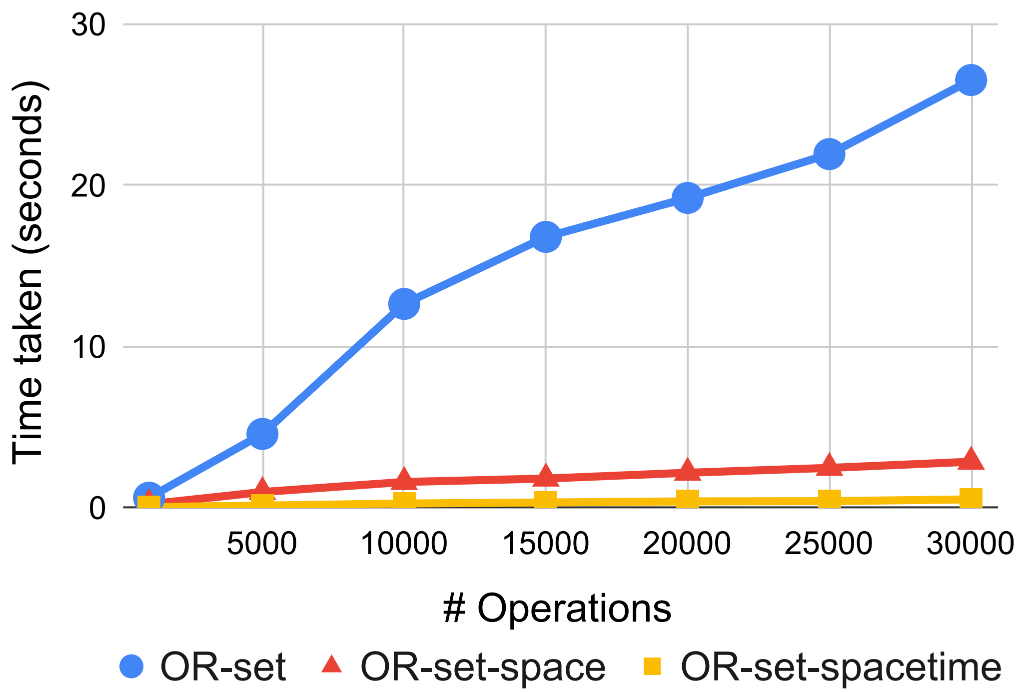
- $AddRem(I) : \forall e \in I.E.\ I.oper(e) = dequeue \wedge$ $I.rval(e) \neq \mathtt{EMPTY} \implies \exists e' \in I.E.\ \mathrm{match}_I(e', e)$
- $Empty(I) : \forall e_1, e_2, e_3 \in I.E.\ I.oper(e_1) = dequeue \wedge$ $I.rval(e_1) = \mathtt{EMPTY}\ \wedge\ I.oper(e_2) = enqueue(a) \wedge$ $e_2 \xrightarrow{I.vis} e_1 \implies \exists e_3 \in I.E.\ \mathrm{match}_I(e_2, e_3) \wedge e_3 \xrightarrow{I.vis} e_1$
- $FIFO_1(I) : \forall e_1, e_2, e_3 \in I.E.\ I.oper(e_1) = enqueue(a) \wedge$ $\mathrm{match}_I(e_2, e_3) \wedge e_1 \xrightarrow{I.vis} e_2 \implies \exists e_4 \in I.E.\ \mathrm{match}_I(e_1, e_4)$
- $FIFO_2(I)$: $\forall e_1, e_2, e_3, e_4 \in I.E.\ \neg(\mathrm{match}_I(e_1, e_4) \wedge$ $\mathrm{match}_I(e_2, e_3)\ \wedge\ e_1 \xrightarrow{I.vis} e_2\ \wedge\ e_3 \xrightarrow{I.vis} e_4)$

# Merge performance of Peepul and Quark[1] Queues




1. *Kaki et. al. Mergeable Replicated Data Types. OOPSLA 19*

# Performance of different OR-Sets



# Compositionality

- Generic $\alpha$-map which can be instantiated with any element type $\alpha$.
- Specification of $\alpha$-map uses the specification of $\alpha$ applied to every key.
- We prove the correctness of $\alpha$-map assuming the correctness of $\alpha$.
- We get a whole family of verified map MRDTs!

$$\mathcal{F}_{\alpha-map}(get(k, o_\alpha), I) = \\ \text{let } I_\alpha = project(k, I) \text{ in } \mathcal{F}_\alpha(o_\alpha, I_\alpha)$$

$\mathcal{D}_{\alpha-map} = (\Sigma, \sigma_0, do, merge_{\alpha-map})$ where

$$\Sigma_{\alpha-map} = \mathcal{P}(string \times \Sigma_\alpha)$$

$$\sigma_0 = \{\}$$

$$\delta(\sigma, k) = \begin{cases} \sigma(k), & \text{if } k \in dom(\sigma) \\ \sigma_{0_\alpha}, & \text{otherwise} \end{cases}$$

$$do(set(k, o_\alpha), \sigma, t) = \\ \text{let } (v, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma[k \mapsto v], r)$$

$$do(get(k, o_\alpha), \sigma, t) = \\ \text{let } (\_, r) = do_\alpha(o_\alpha, \delta(\sigma, k), t) \text{ in } (\sigma, r)$$

$$merge_{\alpha-map}(\sigma_{lca}, \sigma_a, \sigma_b) = \\ \{(k, v) \mid (k \in dom(\sigma_{lca}) \cup dom(\sigma_a) \cup dom(\sigma_b)) \wedge \\ v = merge_\alpha(\delta(\sigma_{lca}, k), \delta(\sigma_a, k), \delta(\sigma_b, k))\}$$

$$\mathcal{R}_{sim-\alpha-map}(I, \sigma) \iff \forall k. \\ (k \in dom(\sigma) \iff \exists e \in I.E.\ oper(e) = set(k, \_)) \wedge \\ \mathcal{R}_{sim-\alpha}\ (project(k, I),\ \delta(\sigma, k))$$

# Conclusion and Future Work

- We have proposed a technique to verify both the functional correctness and convergence of MRDTs.
- We have successfully applied our technique on a number of challenging MRDTs.
- Our technique supports verification of efficient implementations, as well as compositionality through parametric polymorphism.
- Future work: Applying our technique on more complex MRDTs (e.g. JSON Automerge MRDT)
- Future work: Improve automation

# Thank You

Questions?