Modular Verification of Op-Based CRDTs in Separation Logic

Abel Nieto, Léon Gondelman, Alban Reynaud, Amin Timany, Lars Birkedal
Aarhus University
CAP Theorem

- Informally: no distributed data store can be all of the following: (strongly) Consistent, Available, and Partition tolerant.

- Often presented as “choose 2 out of 3”, but some pairs don’t make sense. Additionally, given enough time partitions are unavoidable.

- Better phrasing: given a network partition, your system can be (strongly) consistent or available, but not both.
(Strong) Eventual Consistency

- SEC trades consistency in favour of availability.
- Replica states are allowed to diverge, but must eventually converge.
- Eventual delivery: updates eventually reach all correct replicas.
- Convergence: replicas that have delivered the same updates must be in equivalent states.
- CRDTs: a class of distributed data structures with SEC.
State-based CRDTs

• Updates communicated by sending entire state to other replicas.
• States taken from join semi-lattice, and “merging” states is taking their LUBs.
• Cons: encoding of data type semantics into lattice state can be tricky, inefficient if state is large (but there are pros too)
Op-based CRDTs

- Updates communicated by sending individual operations to other replicas
- Simpler design, but requires exactly-once delivery of operations
- Multiple things can go wrong: message dropped or duplicated, or replica dies
This Work

- We have implemented in OCaml and verified in Aneris a framework for building op-based CRDTs
- We used the framework to implement 12 example CRDTs, including higher-order combinators
- Our specifications are the first to be both modular and about runnable implementations (as opposed to protocols)
- For the first time, our formalisation of CRDTs includes a general-purpose library for Reliable Causal Broadcast (an exactly-once delivery protocol)
CRDT implementor

Purely functional sequential data structure
Coherent with LTS model

LTS model
Coherent with denotation

CRDT denotation

Sample CRDT client

CRDTs
PN-Counter, LWW-Register, AW-Set, ...

OpLib library
RCB properties + convergence, functional correctness

RcbLib library
Causal delivery, no duplication, no creation

Sample RCB client

Aneris logic
Causal Broadcast

- Interface: `init(addrs)`, `broadcast(msg)`, `deliver()`
- Guarantees: no duplication, no creation, and causal delivery
- Causal delivery: for any message $m_1$ that potentially caused $m_2$ (i.e. $m_1 \rightarrow m_2$) then every node delivers $m_1$ before delivering $m_2$
Resources for Causality

• Piggyback on Gondelman et al. (POPL’20): a causally-consistent key-value store

• Locally, track set of messages delivered at replica i: OwnLocal(i, s)

• Globally, track set of all messages ever sent: OwnGlobal(h)

• We can then prove resource laws: e.g. causality

\[
\text{GlobalInv}^{N_{GI}} * \text{OwnLocal}(i, s) * \text{OwnGlobalSnapshot}(h) \vdash \Rightarrow \forall a \in s, w \in h. \, \nu c(w) < \nu c(a) \Rightarrow \exists a' \in s. \, [a'] = w
\]
• The resources also allow us to give specifications to the broadcast and deliver functions

• Simplified broadcast spec:
  \[
  \{\text{OwnGlobal}(h) \ast \text{OwnLocal}(i, s)\}
  \]
  \[
  \langle ip_i; \text{broadcast}(p) \rangle
  \]
  \[
  \{m. \text{payload}(m) = p \ast \text{OwnGlobal}(h \uplus \{m\}) \ast \text{OwnLocal}(i, s \uplus \{m\})\}\]
From Purely-Functional to CRDT

- We start with a purely-functional counter: an initial state (0) and a function to get from a state to the next (effect(c, n) = c + n)

- To turn it into a CRDT we need:

  - A way to propagate operations (RCB).
  - A way to (concurrently) apply remote operations.
  - A way to manage mutable state (because of the above).
OpLib

- A library for implementing operation-based CRDTs
- User (CRDT implementor) provides initial state and effect function
- They get back a fully-fledged CRDT
let effect msg counter = 
    let ((delta, _x), _y) = msg in 
    counter + delta

let init_st () = 0

let crdt = fun () -> (init_st, effect)

let init addrs rid = 
    let initRes = oplib_init int_ser int_deser addrs rid crdt in 
    let (get_state, update) = initRes in 
    (get_state, update)
Specifying OpLib

• Challenge: the CRDT’s current state (e.g. the value of the counter) depends not just on local operations, but also remote ones.

• Tracking current state (Timany et al. 2021):

  \[
  \text{IncrSpec} \\
  \{ \text{gcounter}(i, k) \} \\
  \langle i pi; \text{incr()} \rangle \\
  \{()\}. \exists m. k < m * \text{gcounter}(i, m) \\
  \]

• Solution: don’t track the current state. Instead, track local events \textit{precisely} and a \textit{lower bound} of remote events:

  \[
  \text{LocSt}(i, \bullet s, \circ h)
  \]
Denotations

- In general, can specify CRDT with a denotation: partial function from set of events (including causality data) to CRDT state

- Example (multi-value register)

\[
[s]_{\text{mv-reg}} = \{(w, vc) | \exists o. (\text{write}(w), vc, o) \in s \land vc \in \text{Maximals}(s)\}
\]

- Introduced in Burckhardt et al. [POPL’14] but now adapted to SL
OpLib specs

\textbf{GETSTATESPEC}
\begin{align*}
\langle \text{LocSt}(i, \bullet s, \circ h) \rangle \\
\langle ip_i; \text{get\_state()} \rangle \\
\forall v. \exists h'. w. h' \supseteq h \ast \text{StCoh}(w, v) \ast \\
\text{LocSt}(i, \bullet s, \circ h') \ast \llbracket s \cup h' \rrbracket = w
\end{align*}

\text{Convergence}

\textbf{UPDATESPEC}
\begin{align*}
\langle \text{LocSt}(i, \bullet s, \circ r) \ast \text{GlobSt}(h) \rangle \\
\langle ip_i; \text{update}(v) \rangle \\
\forall a. \exists r'. r' \supseteq r \ast a \notin s \ast a \notin h \ast \text{payload}(a) = v \ast \\
\text{origin}(a) = i \ast a \in \text{Maximals}(h \cup \{a\}) \\
a \in \text{Maximum}(s \cup r' \cup \{a\}) \\
\text{LocSt}(i, \bullet s \cup \{a\}, \circ r') \ast \text{GlobSt}(h \cup \{a\})
\end{align*}
Labelled Transition Systems

• Needed: a way to connect effect function to denotation

• Done via labelled transition system (St, Event, → , σ₀)

• Coherence property:

  1. \([\emptyset] = σ₀\)

  2. \(∀ s \ p \ e \ p'. \text{Valid}(s, e) ∧ [s] = p ∧ p \overset{e}{→} p' \implies [s ∪ e] = p'\)

• Hoare triple for showing that effect() implements LTS
OpLib recap

Sample CRDT client

CRDTs
PN-Counter, LWW-Register, AW-Set, ...

OpLib library
RCB properties + convergence, functional correctness

RcbLib library
Causal delivery, no duplication, no creation

Aneris logic
# Implemented CRDTs

<table>
<thead>
<tr>
<th>CRDT</th>
<th># lines of OCaml</th>
<th># lines of Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive-Negative Counter</td>
<td>25</td>
<td>235</td>
</tr>
<tr>
<td>Grown-only Counter</td>
<td>26</td>
<td>243</td>
</tr>
<tr>
<td>Two-Part Set</td>
<td>25</td>
<td>182</td>
</tr>
<tr>
<td>Add-Wins Set</td>
<td>34</td>
<td>371</td>
</tr>
<tr>
<td>Remove-Wins Set</td>
<td>53</td>
<td>527</td>
</tr>
<tr>
<td>Grow-Only Set</td>
<td>22</td>
<td>159</td>
</tr>
<tr>
<td>Last-Writer-Wins Register</td>
<td>54</td>
<td>555</td>
</tr>
<tr>
<td>Multi-Value Register</td>
<td>35</td>
<td>334</td>
</tr>
<tr>
<td>Product Combinator</td>
<td>30</td>
<td>374</td>
</tr>
<tr>
<td>Map Combinator</td>
<td>34</td>
<td>531</td>
</tr>
<tr>
<td>Table of Positive-Negative Counters</td>
<td>22</td>
<td>74</td>
</tr>
<tr>
<td>Table of Last-Writer-Wins Registers</td>
<td>22</td>
<td>74</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>360</strong></td>
<td><strong>3585</strong></td>
</tr>
</tbody>
</table>

### Libraries

<table>
<thead>
<tr>
<th>Library</th>
<th># lines of OCaml</th>
<th># lines of Coq</th>
</tr>
</thead>
<tbody>
<tr>
<td>RcbLib</td>
<td>196</td>
<td>5019</td>
</tr>
<tr>
<td>OpLib</td>
<td>86</td>
<td>3595</td>
</tr>
<tr>
<td><strong>total</strong></td>
<td><strong>282</strong></td>
<td><strong>8614</strong></td>
</tr>
</tbody>
</table>
Conclusions

• We implemented in OCaml and verified in Aneris a framework for building op-based CRDTs, as well as many examples on top of it

• Ours is the first foundational proof of functional correctness and SEC for op-based CRDTs, as well as the first technique that is both modular and about implementations

• Future work: more complex CRDTs (collaborative text editing) and CRDTs with coordination

• Future work: state-based CRDTs
Thank you