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

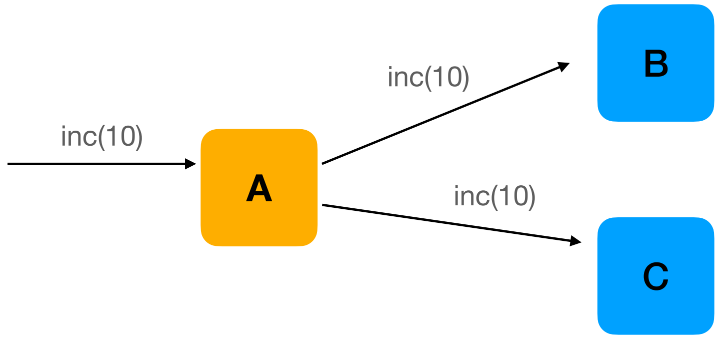
- SEC trades consistency in favour of availability.
- Replica states are allowed to diverge, but must eventually converge.
- Eventual delivery: updates eventually reach all correct replicas.
- Convergence: replicas that have delivered the same updates must be in equivalent states.
- CRDTs: a class of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

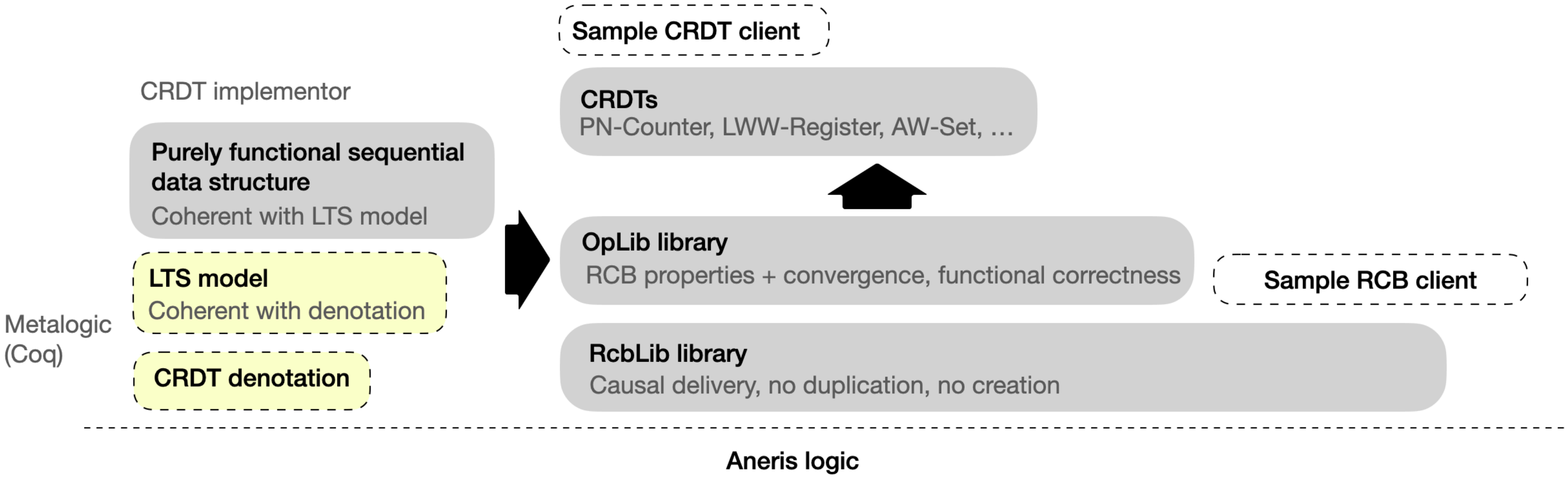
Op-based CRDTs

- 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)



Sample CRDT client
CRDT implementor
CRDTs
PN-Counter, LWW-Register, AW-Set, …
Purely functional sequential data structure
Coherent with LTS model
OpLib library
RCB properties + convergence, functional correctness
Sample RCB client
LTS model
Coherent with denotation
Metalogic (Coq)
CRDT denotation
RcbLib library
Causal delivery, no duplication, no creation
Aneris logic

Causal Broadcast

- Interface: init(addrs), broadcast(msg), deliver()
- Guarantees: no duplication, no creation, and causal delivery
- Causal delivery: for any message m1 that potentially caused m2 (i.e. $m_1 \rightarrow m_2$) then every node delivers m1 before delivering m2

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

$$\boxed{\text{GlobalInv}}^{\mathcal{N}_{\text{GI}}} * \text{OwnLocal}(i, s) * \text{OwnGlobalSnapshot}(h) \vdash \Rrightarrow_{\mathcal{E}} \forall a \in s, w \in h.\ vc(w) < vc(a) \Rightarrow \exists a' \in s.\ \lfloor a' \rfloor = w$$

-

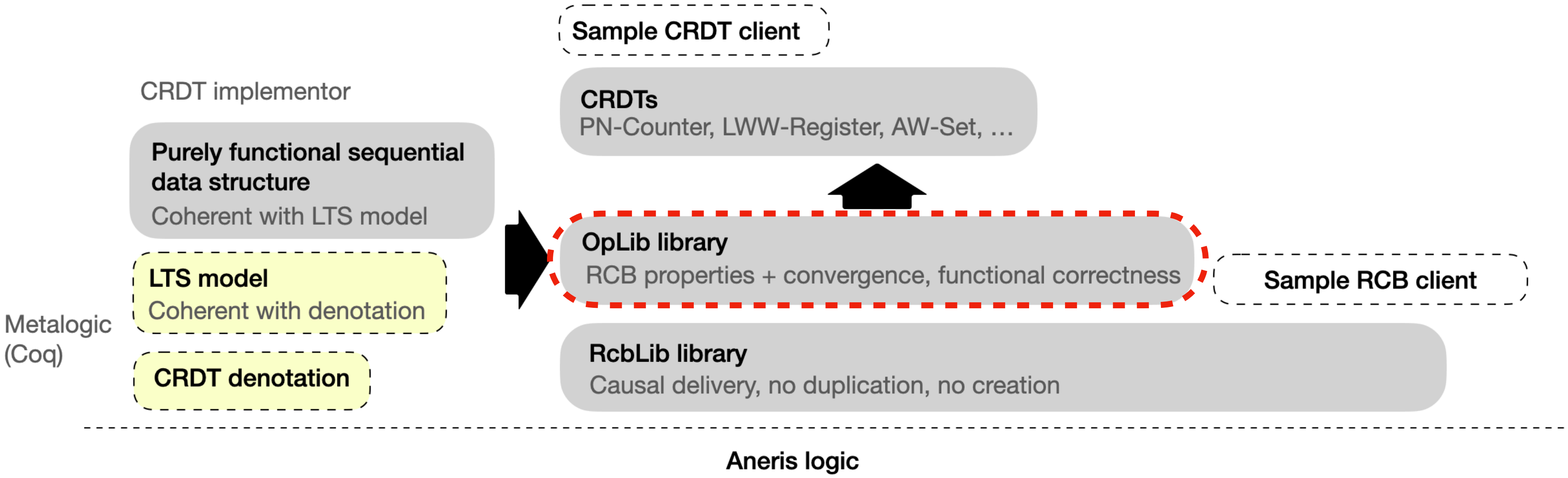
- The resources also allow us to give specifications to the broadcast and deliver functions

- Simplified broadcast spec:

$$\{\text{OwnGlobal}(h) * \text{OwnLocal}(i, s)\}$$

$$\langle ip_i; \text{broadcast}(p) \rangle$$

$$\{m.\ \mathit{payload}(m) = p * \text{OwnGlobal}(h \uplus \{m\}) * \text{OwnLocal}(i, s \uplus \{m\})\}$$


Sample CRDT client
CRDT implementor
CRDTs
PN-Counter, LWW-Register, AW-Set, …
Purely functional sequential data structure
Coherent with LTS model
OpLib library
RCB properties + convergence, functional correctness
Sample RCB client
LTS model
Coherent with denotation
Metalogic (Coq)
CRDT denotation
RcbLib library
Causal delivery, no duplication, no creation
Aneris logic

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):

  IncrSpec

  $\{\mathsf{gcounter}(i, k)\}$

  $\langle ip_i;\ \texttt{incr()} \rangle$

  $\{(). \exists m. k < m * \mathsf{gcounter}(i, m)\}$

- Solution: don't track the current state. Instead, track local events *precisely* and a *lower bound* of remote events:
- $\mathrm{LocSt}(i, \bullet s, \circ h)$

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)
- $$[\![s]\!]_{\mathsf{mv-reg}} = \{(w, vc) | \exists o.(\mathrm{write}(w), vc, o) \in s \wedge vc \in \mathrm{Maximals}(s)\}$$
- Introduced in Burckhardt et al. [POPL'14] but now adapted to SL

OpLib specs

GetStateSpec

$$\langle \mathrm{LocSt}(i, \bullet\, s, \circ\, h) \rangle$$

$$\langle ip_i;\ \mathrm{get\_state}() \rangle$$

$$\left\langle \begin{array}{l} v.\ \exists h'\, w.\ h' \supseteq h * \mathrm{StCoh}(w, v) * \\ \quad \mathrm{LocSt}(i, \bullet\, s, \circ\, h') * [\![s \cup h']\!] = w \end{array} \right\rangle^{\mathcal{N}}$$

Convergence

UpdateSpec

$$\langle \mathrm{LocSt}(i, \bullet\, s, \circ\, r) * \mathrm{GlobSt}(h) \rangle$$

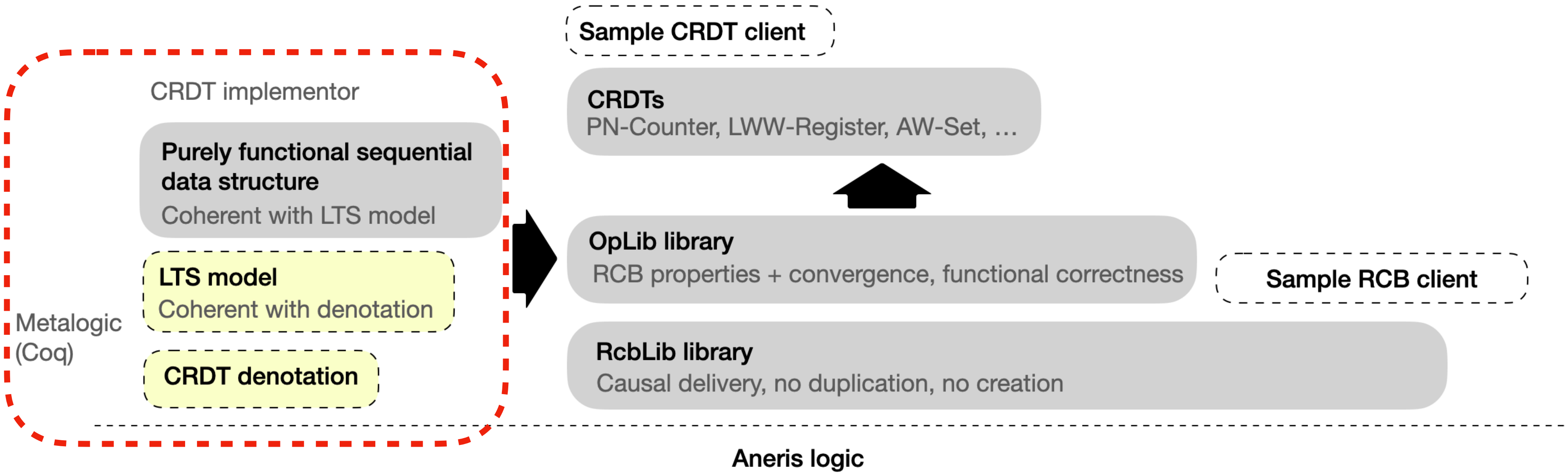
$$\langle ip_i;\ \mathrm{update}(v) \rangle$$

$$\left\langle \begin{array}{l} ().\ \exists a\, r'.\ r' \supseteq r * a \notin s * a \notin h * \mathit{payload}(a) = v * \\ \quad \mathit{origin}(a) = i * a \in \mathrm{Maximals}(h \cup \{a\}) * \\ \quad a \in \mathrm{Maximum}(s \cup r' \cup \{a\}) * \\ \quad \mathrm{LocSt}(i, \bullet\, s \cup \{a\}, \circ\, r') * \mathrm{GlobSt}(h \cup \{a\}) \end{array} \right\rangle^{\mathcal{N}}$$

Labelled Transition Systems

- Needed: a way to connect effect function to denotation
- Done via labelled transition system $(\text{St}, \text{Event}, \rightarrow, \sigma_0)$
- Coherence property:
- $[\![\emptyset]\!] = \sigma_0$
- $\forall s\, p\, e\, p'.\text{Valid}(s, e) \wedge [\![s]\!] = p \wedge p \xrightarrow{e} p' \implies [\![s \cup e]\!] = p'$
- Hoare triple for showing that effect() implements LTS

OpLib recap



Sample CRDT client
CRDT implementor
Purely functional sequential data structure
Coherent with LTS model
LTS model
Coherent with denotation
Metalogic (Coq)
CRDT denotation
CRDTs
PN-Counter, LWW-Register, AW-Set, …
OpLib library
RCB properties + convergence, functional correctness
Sample RCB client
RcbLib library
Causal delivery, no duplication, no creation
Aneris logic

Implemented CRDTs

CRDT	# lines of OCaml	# lines of Coq	
Positive-Negative Counter	25	235	Simple CRDTs
Grown-only Counter	26	243	Simple CRDTs
Two-Part Set	25	182	Simple CRDTs
Add-Wins Set	34	371	Simple CRDTs
Remove-Wins Set	53	527	Simple CRDTs
Grow-Only Set	22	159	Simple CRDTs
Last-Writer-Wins Register	54	555	Simple CRDTs
Multi-Value Register	35	334	Simple CRDTs
Product Combinator	30	374	Combinators
Map Combinator	34	531	Combinators
Table of Positive-Negative Counters	22	74	Compound CRDTs
Table of Last-Writer-Wins Registers	22	74	Compound CRDTs
total	360	3585	

combined to make

combined to make

Library	# lines of OCaml	# lines of Coq
RcbLib	196	5019
OpLib	86	3595
total	282	8614

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