# Composition of State-based CRDTs

Carlos Baquero, Paulo Sérgio Almeida, Alcino Cunha and Carla Ferreira

May 25, 2015

## 1 Introduction

State-based CRDTs are rooted in mathematical structures called join-semilattices (ore simply lattices, in this context). These order structures ensure that the replicated states of the defined data types evolve and increase in a partial order in a sufficiently defined way, so as to ensure that all concurrent evolutions can be merged deterministically. In order to build, or understand the building principles, of state-based CRDTs it is necessary to understand the basic building blocks of the support lattices and how lattices can be composed.

## 2 From Sets to Lattices

In this context the most basic structure to define is a $\mathsf{set}$ of distinct values. An example is the set of vowels that can defined by extension as $\mathsf{vowels} \doteq \{a, e, i, o, u\}$. Elements in a set have no specific order and they only need to be distinguishable, i.e. by defining $=$.

Having a $\mathsf{set}$ we can define partial orders by defining a $\mathsf{poset}$ over a support $\mathsf{set}$ and an order relation $\sqsubseteq$. This relation can be any binary relation that is reflexive, transitive and anti-symmetric. Given elements $o, p, q$ in a set.

- (reflexive) $o \sqsubseteq o$
- (transitive) $o \sqsubseteq p \wedge p \sqsubseteq q \Rightarrow o \sqsubseteq q$
- (anti-symmetric) $o \sqsubseteq p \wedge p \sqsubseteq o \Rightarrow o = p$

Since sets already define $=$ it is possible to create posets transitively by enumerating the element pairs related by $\sqsubset$. As an example, we can build a $\mathsf{poset}$ with a total order on the set of vowels by defining $\langle \mathsf{vowels}, \{(a, e), (e, i), (i, o), (o, u)\}\rangle$ In this example we ordered all elements and thus created a $\mathsf{chain}$, with $a \sqsubset e \sqsubset i \sqsubset o \sqsubset u$, i.e. given any two elements $o, p$ either $o \sqsubseteq p$ or $p \sqsubseteq o$.

If some elements were left unordered we could have concurrent elements.

- (concurrent) $o \parallel p \iff \neg(o \sqsubseteq p \vee p \sqsubseteq o)$

In the extreme case we could have left all elements unordered and defined a $\mathsf{poset}$ that depicted an *antichain*, where any two elements are always concurrent. E.g. $\langle \mathsf{vowels}, \{\}\rangle$. Having a $\mathsf{poset}$ we also have the properties of a $\mathsf{set}$.

$$\frac{A : \mathsf{poset}}{A : \mathsf{set}}$$

For a given $\mathsf{poset}$ to be a join-semilattice there must be a *least-upper-bound* for any subset of the support set. Given a pair of elements $o, p$, their least-upper-bound can be derived by the result of a binary join operator, by $o \sqcup p$. Since the binary join is commutative and associative it can be iterated over the elements of any subset to derive the least-upper-bound of the subset. Some properties of join are listed bellow.

- (idempotent) $o \sqcup o = o$
- (commutative) $o \sqcup p = p \sqcup o$
- (associative) $o \sqcup (p \sqcup q) = (o \sqcup p) \sqcup q$

And properties of least-upper-bounds.

- (upper-bound) $o \sqsubseteq o \sqcup p$
- (least-upper-bound) $o \sqsubseteq q \wedge o \sqsubseteq q \Rightarrow o \sqsubseteq p \sqcup q$

A general example of a $\mathsf{poset}$ with a join is obtained from any $\mathsf{set}$ by selecting the order to be set inclusion and the join to be set union. In our running example this would be the $\mathsf{lattice}$ defined by $\langle \mathsf{vowels}, \subseteq, \cup\rangle$. Another simple lattice can be obtained by taking the maximum in a total order (or dually the minimum), for naturals we can derive $\mathsf{maxint} \doteq \langle \mathbb{N}, \leq_{\mathbb{N}}, \mathbf{max}\rangle$.

Having a $\mathsf{lattice}$ we also have the properties of a $\mathsf{poset}$.

$$\frac{A : \mathsf{lattice}}{A : \mathsf{poset}}$$

A $\mathsf{chain}$ (a special case of a $\mathsf{poset}$) always derives a $\mathsf{lattice}$.

$$\frac{A : \mathsf{chain}}{A : \mathsf{lattice}}$$

Notice that although some specific partial orders always derive lattices, as is the case for *chains*, in general we can have partial orders that are not lattices. An example is the prefix ordering on bit strings that can produce concurrent elements, $010 \parallel 100$, and is not a lattice.

We will see in latter sections that in some cases it is useful to have a special element in the lattice that is the bottom element $\bot$. Some properties are.

- (bottom) $\bot \sqsubseteq o$
- (identity) $\bot \sqcup o = o$

The lattice formed by set inclusion has the empty set as bottom, $\langle \mathsf{vowels}, \subseteq, \cup, \emptyset \rangle$. Not all lattices have a "natural" bottom, but it is always possible to add an extra element as bottom to an existing lattice. We will address this construction when talking about lattice composition by linear sums. As expected, lattices with bottom also have the lattice properties.

$$\frac{A : \mathsf{lattice}_\bot}{A : \mathsf{lattice}}$$

## 2.1 Primitive Lattices

We now introduce a small set of lattices, that will be later useful to construct more complex structures by composition.

**Singleton** A single element, $\bot$.

$$\frac{}{\mathbb{1} : \mathsf{lattice}_\bot}$$

$$\bot \sqsubseteq \bot \qquad \bot \sqcup \bot = \bot$$

**Boolean** Two elements $\mathbb{B} = \{\mathsf{False}, \mathsf{True}\}$ in a chain, join is logical $\vee$.

$$\frac{}{\mathbb{B} : \mathsf{lattice}_\bot}$$

$$\mathsf{False} \sqsubseteq \mathsf{True} \qquad x \sqcup y = x \vee y \qquad \bot = \mathsf{False}$$

**Naturals** Natural numbers. We include the 0, thus $\mathbb{N} = \{0, 1, \ldots\}$.

$$\frac{}{\mathbb{N} : \mathsf{lattice}_\bot}$$

$$n \sqsubseteq m = n \leq m \qquad n \sqcup m = \max(n, m) \qquad \bot = 0$$

## 3 Inflations make CRDTs

State-based CRDTs can be specified by selecting a given lattice to model the state, and choosing an initial value in the lattice, usually the $\bot$. Mutation operations can only change the state by *inflations* and do not return values. Query operations evaluate an arbitrary function on the state and return a value.

An inflation is an endo-function on the lattice type that picks a value $x$ among the set of valid lattice states $a$ and produces a new value state such that:

- (inflation) $x \sqsubseteq f(x)$

Inflations can be further classified as non-strict and strict inflations, where a strict inflation is such that:

- (strict inflation) $x \sqsubset f(x)$

We can now classify inflations.

$$\frac{\forall x \in a \cdot x \sqsubseteq f(x)}{f : A \xrightarrow{\sqsubseteq} A}$$

$$\frac{\forall x \in a \cdot x \sqsubset f(x)}{f : A \xrightarrow{\sqsubset} A}$$

$$\frac{f : A \xrightarrow{\sqsubset} A}{f : A \xrightarrow{\sqsubseteq} A}$$

A state that is only updated as a result of an inflation over its current value, is immutable under joins with copies of past states.

Notice that an inflation is not the same as a monotonic function, $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$. Example, the function $f(x) = \frac{x}{2}$ on positive reals is monotonic and is not an inflation.

## 3.1 Primitive Inflations

Building on the primitive lattices introduced above we can build some inflations.

$$\mathsf{id}(x) = x \qquad \frac{}{\mathsf{id} : A \xrightarrow{\sqsubseteq} A}$$

$$\underline{\mathsf{True}}(x) = \mathsf{True} \qquad \frac{}{\underline{\mathsf{True}} : \mathbb{B} \xrightarrow{\sqsubseteq} \mathbb{B}}$$

$$\mathsf{succ}(x) = x + 1 \qquad \frac{}{\mathsf{succ} : \mathbb{N} \xrightarrow{\sqsubset} \mathbb{N}}$$

## 3.2 Sequential Composition

Inflations can be composed sequentially. As long as there is at least one strict inflation in the composition, we are sure to also have a strict composition.

$$(f \bullet g)(x) = f(g(x))$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : A \xrightarrow{\sqsubseteq} A}{f \bullet g : A \xrightarrow{\sqsubseteq} A}$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : A \xrightarrow{\sqsubset} A}{f \bullet g : A \xrightarrow{\sqsubset} A} \qquad \frac{f : A \xrightarrow{\sqsubset} A \quad g : A \xrightarrow{\sqsubseteq} A}{f \bullet g : A \xrightarrow{\sqsubset} A}$$

## 4 Lattice Compositions

Since we are interested in creating lattices we consider a few composition techniques that are known to derive lattices. While in some cases they build from other lattices, in others they can derive lattices from simpler structures.

### 4.1 Product

The product $\times$, or pair construction, derives a lattice formed by pairs of other lattices. It can be applied recursively and derive a composition from a sequence of lattices, where operations are applied in point-wise order.

$$\frac{A : \mathsf{lattice} \qquad B : \mathsf{lattice}}{A \times B : \mathsf{lattice}}$$

$$(x_1, y_1) \sqsubseteq (x_2, y_2) = x_1 \sqsubseteq x_2 \wedge y_1 \sqsubseteq y_2$$

$$(x_1, y_1) \sqcup (x_2, y_2) = (x_1 \sqcup x_2, y_1 \sqcup y_2)$$

The construction also extends to $\mathsf{lattice}_\bot$ when all sources are also $\mathsf{lattice}_\bot$.

$$\frac{A : \mathsf{lattice}_\bot \qquad B : \mathsf{lattice}_\bot}{A \times B : \mathsf{lattice}_\bot}$$

$$\bot = (\bot, \bot)$$

As an example, the underlying lattice structure of a version vector among three replica nodes is composable by $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$ with $\bot = (0, 0, 0)$.

Bellow are the properties of inflations over products. A strict inflation on one of the components leads to an overall strict inflation.

$$(f \times g)(x, y) = (f(x), g(y))$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : B \xrightarrow{\sqsubseteq} B}{f \times g : A \times B \xrightarrow{\sqsubseteq} A \times B}$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : B \xrightarrow{\sqsubset} B}{f \times g : A \times B \xrightarrow{\sqsubset} A \times B} \qquad \frac{f : A \xrightarrow{\sqsubset} A \quad g : B \xrightarrow{\sqsubseteq} B}{f \times g : A \times B \xrightarrow{\sqsubset} A \times B}$$

### 4.2 Lexicographic Product

The $\boxtimes$ construct builds a lexicographic order from its source lattices. Components to the left are more significant and, unless they are equal, they filter out further comparisons to the right side.

$$\frac{A : \mathsf{lattice} \qquad B : \mathsf{lattice}_\bot}{A \boxtimes B : \mathsf{lattice}} \qquad \frac{A : \mathsf{lattice}_\bot \qquad B : \mathsf{lattice}_\bot}{A \boxtimes B : \mathsf{lattice}_\bot}$$

$$(x_1, y_1) \sqsubseteq (x_2, y_2) = x_1 \sqsubseteq x_2 \vee (x_1 = x_2 \wedge y_1 \sqsubseteq y_2)$$

$$(x_1, y_1) \sqcup (x_2, y_2) = \begin{cases} (x_1, y_1) & \textbf{if } x_2 \sqsubset x_1 \\ (x_2, y_2) & \textbf{if } x_1 \sqsubset x_2 \\ (x_1, y_1 \sqcup y_2) & \textbf{if } x_1 = x_2 \\ (x_1 \sqcup x_2, \bot) & \textbf{otherwise} \end{cases}$$

$$\bot = (\bot, \bot)$$

In the join definition we can observe that the $\bot$ value is used only when the left components can have concurrent values. Note that $B$ could be simply a lattice ($B$ : $\mathsf{lattice}$) and then join definition could be redefined in the following manner:

$$(x_1, y_1) \sqcup (x_2, y_2) = \begin{cases} (x_1, y_1) & \textbf{if } x_2 \sqsubset x_1 \\ (x_2, y_2) & \textbf{if } x_1 \sqsubset x_2 \\ (x_1, y_1 \sqcup y_2) & \textbf{if } x_1 = x_2 \\ (x_1 \sqcup x_2, y_1 \sqcup y_2) & \textbf{otherwise} \end{cases}$$

If the left component is a $\mathsf{chain}$, often the case in practical uses, then the right one can be a simple $\mathsf{lattice}$ (without $\bot$) and the fourth clause of the join definition is not used.

$$\frac{A : \mathsf{chain} \qquad B : \mathsf{lattice}}{A \boxtimes B : \mathsf{lattice}}$$

And, if the right component is also a $\mathsf{chain}$ the composition is a $\mathsf{chain}$.

$$\frac{A : \mathsf{chain} \qquad B : \mathsf{chain}}{A \boxtimes B : \mathsf{chain}}$$

Properties of inflations.

$$(f \boxtimes g)(x, y) = (f(x), g(y))$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : B \xrightarrow{\sqsubseteq} B}{f \boxtimes g : A \boxtimes B \xrightarrow{\sqsubseteq} A \boxtimes B}$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : B \xrightarrow{\sqsubset} B}{f \boxtimes g : A \boxtimes B \xrightarrow{\sqsubset} A \boxtimes B} \qquad \frac{f : A \xrightarrow{\sqsubset} A \quad g : B \longrightarrow B}{f \boxtimes g : A \boxtimes B \xrightarrow{\sqsubset} A \boxtimes B}$$

Notice that if we apply a strict inflation to the left component, then the right can be transformed by any (endo-)function even if non inflationary. In practice this allows resetting the right component after strictly inflating the left.

## 4.3 Linear Sum

The next composition, linear sum $\oplus$, picks two lattices, left and right, and creates a new lattice where any element from the left lattice is always lower that any element in the right lattice. In the resulting set the elements are tagged with a label that identifies from which source lattice they came form. i.e. $\mathsf{Left}\ a$ means that element $a$ came from the left lattice and is now named $\mathsf{Left}\ a$. Tagging also ensures that the sets supporting each lattice could have had elements in common.

$$\frac{A : \mathsf{lattice} \qquad B : \mathsf{lattice}}{A \oplus B : \mathsf{lattice}} \qquad \frac{A : \mathsf{lattice}_\bot \qquad B : \mathsf{lattice}}{A \oplus B : \mathsf{lattice}_\bot}$$

$$\begin{aligned}
\mathsf{Left}\ x &\sqsubseteq \mathsf{Left}\ y &&= x \sqsubseteq y & \mathsf{Left}\ x &\sqcup \mathsf{Left}\ y &&= \mathsf{Left}\ (x \sqcup y)\\
\mathsf{Right}\ x &\sqsubseteq \mathsf{Right}\ y &&= x \sqsubseteq y & \mathsf{Right}\ x &\sqcup \mathsf{Right}\ y &&= \mathsf{Right}\ (x \sqcup y)\\
\mathsf{Left}\ x &\sqsubseteq \mathsf{Right}\ y &&= \mathsf{True} & \mathsf{Left}\ x &\sqcup \mathsf{Right}\ y &&= \mathsf{Right}\ y\\
\mathsf{Right}\ x &\sqsubseteq \mathsf{Left}\ y &&= \mathsf{False} & \mathsf{Right}\ x &\sqcup \mathsf{Left}\ y &&= \mathsf{Right}\ x
\end{aligned}$$

$$\bot = \mathsf{Left}\ \bot$$

A possible use of this construction is to add a $\bot$ to a lattice that didn't had one. For instance $\mathbb{1} \oplus \mathbb{R}$ can add a special element, e.g. $\mathsf{nil}$, that is ordered as lower than any real number. The same construction can also be used to add a top element $\top$ to a lattice, that can act as a tombstone that stops lattice evolution. Notice that for any state $x$, $x \sqcup \top = \top$.

Properties of inflations.

$$\begin{aligned}
(f \oplus g)(\mathsf{Left}\ x) &= \mathsf{Left}\ f(x)\\
(f \oplus g)(\mathsf{Right}\ x) &= \mathsf{Right}\ g(x)
\end{aligned}$$

$$\frac{f : A \xrightarrow{\sqsubseteq} A \quad g : B \xrightarrow{\sqsubseteq} B}{f \oplus g : A \oplus B \xrightarrow{\sqsubseteq} A \oplus B}$$

$$\frac{f : A \xrightarrow{\sqsubset} A \quad g : B \xrightarrow{\sqsubset} B}{f \oplus g : A \oplus B \xrightarrow{\sqsubset} A \oplus B}$$

## 4.4 Function and Map

A total function $\rightarrow$ is obtained by combining a $\mathsf{set}$ with a $\mathsf{lattice}$. This construction does keywise comparison and joins.

$$\frac{A : \mathsf{set} \qquad B : \mathsf{lattice}}{A \rightarrow B : \mathsf{lattice}} \qquad \frac{A : \mathsf{set} \qquad B : \mathsf{lattice}_\bot}{A \rightarrow B : \mathsf{lattice}_\bot}$$

$$f \sqsubseteq g = \forall x \in A \cdot f(x) \sqsubseteq g(x) \qquad (f \sqcup g)(x) = f(x) \sqcup g(x)$$

$$\bot(x) = \bot$$

A map $\hookrightarrow$ can be obtained from a function by assigning a bottom to keys that are not present in a given map, and then using the function definitions. The linear sum construction is used to assign a distinguished bottom to any $\mathsf{lattice}$ $V$ in the co-domain.

$$K \hookrightarrow V \cong K \to \mathbb{1} \oplus V$$

$$\frac{K : \mathsf{set} \qquad V : \mathsf{lattice}}{K \hookrightarrow V : \mathsf{lattice}_\bot}$$

For example, we can define a map of $\mathsf{vowels}$ keys to integer counters $\mathsf{vowels} \hookrightarrow \mathbb{N}$ by using a total function $\mathsf{vowels} \to \mathbb{1} \oplus \mathbb{N}$. Where the map state $\{a \mapsto 3, i \mapsto 5\}$ would be the same as the function state $\{a \mapsto 3, e \mapsto \bot, i \mapsto 5, o \mapsto \bot, u \mapsto \bot\}$.

We define some inflations over maps. The first inflation applies an inflation to all values in the co-domain and thus inflates the map composition.

$$\mathsf{map}(f)(m) = \{(k, f(v)) \mid (k, v) \in m\}$$

$$\frac{f : V \xrightarrow{\sqsubseteq} V}{\mathsf{map}(f) : (K \hookrightarrow V) \xrightarrow{\sqsubseteq} (K \hookrightarrow V)}$$

The second inflation transforms the value on a given key, and if the key is missing applies it to $\bot$. This allows a strict inflation in the co-domain lattice to imply a strict inflation in the composition.

$$\mathsf{apply}_k(f)(m) = \begin{cases} m\{k \mapsto f(v)\} & \textbf{if } (k, v) \in m \\ m\{k \mapsto f(\bot)\} & \textbf{otherwise} \end{cases}$$

$$\frac{f : V \xrightarrow{\sqsubseteq} V}{\mathsf{apply}_k(f) : (K \hookrightarrow V) \xrightarrow{\sqsubseteq} (K \hookrightarrow V)}$$

$$\frac{f : V \xrightarrow{\sqsubset} V}{\mathsf{apply}_k(f) : (K \hookrightarrow V) \xrightarrow{\sqsubset} (K \hookrightarrow V)}$$

## 4.5 Sets and Multisets

Given any $\mathsf{set}$ $A$ it is possible to derive a $\mathsf{lattice}_\bot$ by using the set of all possible subsets, the *powerset* $\mathcal{P}(A)$.

For example, $\mathcal{P}(\{x, y, z\}) = \{\{\}, \{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$.

$$\frac{A : \mathsf{set}}{\mathcal{P}(A) : \mathsf{lattice}_\bot}$$

$$\mathcal{P}(A) \cong A \to \mathbb{B}$$

$$a \sqsubseteq b = a \subseteq b \qquad a \sqcup b = a \cup b \qquad \bot = \{\}$$

The *powerset* can also be represented by a function composition that maps each set element to a boolean that states its presence in the subset.

This composition is very general since it can produce a $\mathsf{lattice}_\bot$ from any $\mathsf{set}$.

A natural extension is to represent *mutisets* by mapping the domain set to naturals, instead of booleans.

$$\frac{A : \mathsf{set}}{\mathcal{M}(A) : \mathsf{lattice}_\bot}$$

$$\mathcal{M}(A) \cong A \to \mathbb{N}$$

$$a \sqsubseteq b = a \subseteq b \qquad a \sqcup b = a \cup b \qquad \bot = \{\}$$

The generic inflations defined for functions when used here show that adding elements is inflationary. For sets represented by $A \to \mathbb{B}$ with a given state $s$ we can define how to add an element $e$.

$$\mathsf{add}(e)(s) = \mathsf{apply}_e(\underline{\mathsf{True}})(s)$$

Likewise, when adding on multisets $A \to \mathbb{N}$ one increments the element count, having a strict inflation.

$$\mathsf{add}(e)(s) = \mathsf{apply}_e(\mathsf{succ})(s)$$

## 4.6 Antichain of Maximal Elements

Starting from a $\mathsf{poset}$ this construction produces a $\mathsf{lattice}_\bot$ by keeping an antichain of maximal elements, given the base $\mathsf{poset}$ order. Upon join, all elements that are concurrent are kept, but any element that is present together with a higher element is removed.

$$\frac{A : \mathsf{poset}}{\mathcal{A}(A) : \mathsf{lattice}_\bot}$$

$$\mathcal{A}(A) = \{\mathsf{maximal}(a) \mid a \in \mathcal{P}(A)\}$$

$$\mathsf{maximal}(a) = \{x \in a \mid \nexists y \in a \cdot x \sqsubset y\}$$

$$a \sqsubseteq b = \forall x \in a \cdot \exists y \in b \cdot x \sqsubseteq y$$

$$a \sqcup b = \mathsf{maximal}(a \cup b)$$

$$\bot = \{\}$$

## 5 Abridged Catalog

In order to exemplify the composition constructs we present a small set of example CRDTs. Simple query functions are included and all mutators are inflations.

Notice that join does not need to be defined as it follows from the composition rules that were introduced.

### 5.1 Positive Counter

This simple form of counter can only increase. Replica nodes must have access to unique ids among a set $I$ and can only increment its position in a map of ids to integers. While increment mutators are parametrized by id $i$ the query is anonymous and simply inspects the state.

$$\mathsf{PCounter} = I \hookrightarrow \mathbb{N}$$

$$\begin{aligned}
\mathsf{inc}_i(a) &= \mathsf{apply}_i(\mathsf{succ})(a) \\
\mathsf{value}(a) &= \sum\{v \mid (i, v) \in a\}
\end{aligned}$$

Notice that if a given node does not yet have an entry in the map and increments, then succ applies over $\perp$, which for $\mathbb{N}$ was defined to be 0.

**Positive counter with reset**

$$\mathsf{PCounter} = (I \hookrightarrow \mathbb{N}) \times (I \hookrightarrow \mathbb{N})$$

$$\begin{aligned}
\mathsf{inc}_i(a) &= \mathsf{apply}_i(\mathsf{succ})(\mathsf{fst}(a)) \\
\mathsf{reset}(a) &= \perp \times \mathsf{fst}(a) \sqcup \mathsf{snd}(a) \\
\mathsf{value}(a) &= \sum\{v \mid (i, v) \in \mathsf{fst}(a)\} - \sum\{v \mid (i, v) \in \mathsf{snd}(a)\}
\end{aligned}$$

### 5.2 Positive and Negative Counter

This variation allows for both increments and decrements. A solution is to pair two positive counters and consider the right side as negative. We use the standard functions fst() and snd() to respectively access the left and right elements of a pair.

$$\mathsf{PNCounter} = I \hookrightarrow \mathbb{N} \times I \hookrightarrow \mathbb{N}$$

$$\begin{aligned}
\mathsf{inc}_i(a) &= \mathsf{apply}_i(\mathsf{succ})(\mathsf{fst}(a)) \\
\mathsf{dec}_i(a) &= \mathsf{apply}_i(\mathsf{succ})(\mathsf{snd}(a)) \\
\mathsf{value}(a) &= \sum\{v \mid (i, v) \in \mathsf{fst}(a)\} - \sum\{v \mid (i, v) \in \mathsf{snd}(a)\}
\end{aligned}$$

An alternative way to obtain a similar result is to use a lexicographic pair and have the first element incremented when one needs to update the count on the second element.

$$\mathsf{PNCCounter} = I \hookrightarrow \mathbb{N} \boxtimes \mathbb{Z}$$

$$\begin{aligned}
\mathsf{inc}_i(a) &= \mathsf{apply}_i(\mathsf{id} \boxtimes \mathsf{succ})(a) \\
\mathsf{dec}_i(a) &= \mathsf{apply}_i(\mathsf{succ} \boxtimes \mathsf{pred})(a) \\
\mathsf{value}(a) &= \sum \{\mathsf{snd}(v) \mid (i, v) \in a\}
\end{aligned}$$

$$\mathsf{pred}(x) = x - 1$$

## 5.3 Observed-remove Add-wins Set

An observed-remove set with add-wins semantics can be derived by creating unique tokens whenever a new element is inserted, using for that a grow only counter per replica, and canceling this tokens, by increasing a boolean to True, upon removal. Only elements supported by non-canceled tokens are considered to be in the set.

$$\mathsf{ORSet}^{+} = E \hookrightarrow I \hookrightarrow \mathbb{N} \boxtimes \mathbb{B}$$

$$\begin{aligned}
\mathsf{add}_{e,i}(a) &= \mathsf{apply}_e(\mathsf{apply}_i(\mathsf{succ} \boxtimes \underline{\mathsf{False}}))(a) \\
\mathsf{rmv}_e(a) &= \mathsf{apply}_e(\mathsf{map}(\mathsf{id} \boxtimes \underline{\mathsf{True}}))(a) \\
\\
\mathsf{member}_e(a) &= \exists (e, m) \in a \cdot \exists i, n \cdot (n, \mathsf{False}) \in m(i)
\end{aligned}$$

## 5.4 Observed-remove Remove-wins Set

An observed-remove set with remove-wins semantics is derived by a dual construction to the previous one, while sharing the same state lattice. Removal creates unique tokens, and additions need to cancel all remove tokens that are visible in the state.

$$\mathsf{ORSet}^{-} = E \hookrightarrow I \hookrightarrow \mathbb{N} \boxtimes \mathbb{B}$$

$$\begin{aligned}\mathsf{rmv}_{e,i}(a) &= \mathsf{apply}_e(\mathsf{apply}_i(\mathsf{succ} \boxtimes \underline{\mathsf{False}}))(a)\\ \mathsf{add}_e(a) &= \mathsf{apply}_e(\mathsf{map}(\mathsf{id} \boxtimes \underline{\mathsf{True}}))(a)\\ \\ \mathsf{member}_e(a) &= \exists (e,m) \in a \cdot \nexists i, n \cdot (n, \mathsf{False}) \in m(i)\end{aligned}$$

## 5.5 Enable-wins Flag

A boolean flag that can be flipped, implemented in Riak under the name flag data type. It is a special case of an $\mathsf{ORSet}^+$ for a singleton set. Flag starts disabled.

$$\mathsf{Flag}^+ = I \hookrightarrow \mathbb{N} \boxtimes \mathbb{B}$$

$$\begin{aligned}\mathsf{enable}_i(a) &= \mathsf{apply}_i(\mathsf{succ} \boxtimes \underline{\mathsf{False}})(a)\\ \mathsf{disable}(a) &= \mathsf{map}(\mathsf{id} \boxtimes \underline{\mathsf{True}})(a)\\ \\ \mathsf{value}(a) &= \exists i, n \cdot (n, \mathsf{False}) \in a(i)\end{aligned}$$

## 5.6 Disable-wins Flag

A boolean flag that can be flipped, implemented in Riak under the name flag data type. It is a special case of an $\mathsf{ORSet}^-$ for a singleton set. Flag starts enabled.

$$\mathsf{Flag}^- = I \hookrightarrow \mathbb{N} \boxtimes \mathbb{B}$$

$$\begin{aligned}\mathsf{disable}_i(a) &= \mathsf{apply}_i(\mathsf{succ} \boxtimes \underline{\mathsf{False}})(a)\\ \mathsf{enable}(a) &= \mathsf{map}(\mathsf{id} \boxtimes \underline{\mathsf{True}})(a)\\ \\ \mathsf{value}(a) &= \nexists\, i, n \cdot (n, \mathsf{False}) \in a(i)\end{aligned}$$

## 5.7 Multi-value Register

A non-optimized multi-value register can be derived by lexicographic coupling of a version vector clock $C$ with a payload value $V$. When a new value $v$ is to be assigned, a new clock, greater than all visible clocks in the state, is created and coupled with the value. These pairs are kept in a antichain of maximal elements. Thus, upon merge, concurrently assigned values will be collected, but any subsequent assignment will again reduce the state to a single pair value.

$$\begin{aligned}\mathsf{MVReg}(V, I) &= \mathcal{A}(C \boxtimes V)\\ C &= I \hookrightarrow \mathbb{N}\end{aligned}$$

$$\begin{aligned}\mathsf{assign}_{v,i}(a) &= \{\mathsf{apply}_i(\mathsf{succ})(\bigsqcup\{c \mid (c, v') \in a\}) \boxtimes v\}\\ \mathsf{values}(a) &= \{v \mid (c, v) \in a\}\end{aligned}$$

Notice that the value is never updated without creating a new clock. Thus, lexicographic comparison (needed for the operation of the antichain join) is always decided by the first component, and in practice $V$ can be any opaque payload without need to define a partial order on its values.

## 6 Closing Remarks

This report collects several composition techniques for lattices, adopts the notion of inflation and how it applies to the specification of state based CRDTs over lattices. Most of the lattice compositions are very standard techniques from order theory [5]. An early version of this work was presented at Schloss Dagstuhl under the title *Composition of Lattices and CRDTs* and the summary of the presentation is available at [6]. Most of the CRDT constructions used here are influenced by work in [8, 7, 2, 4, 3, 1].

## References

[1] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. Making operation-based CRDTs operation-based. In *Proceedings of Distributed Applications and Interoperable Systems: 14th IFIP WG 6.1 International Conference*. Springer, 2014.

[2] Annette Bieniusa, Marek Zawirski, Nuno Preguiça, Marc Shapiro, Carlos Baquero, Valter Balegas, and Sérgio Duarte. Brief announcement: Semantics of eventually consistent replicated sets. In Marcos K. Aguilera, editor, *Int. Symp. on Dist. Comp. (DISC)*, volume 7611 of *Lecture Notes in Comp. Sc.*, pages 441–442, Salvador, Bahia, Brazil, October 2012. Springer-Verlag.

[3] Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: specification, verification, optimality. In

Suresh Jagannathan and Peter Sewell, editors, *POPL*, pages 271–284. ACM, 2014.

[4] Neil Conway, William R Marczak, Peter Alvaro, Joseph M Hellerstein, and David Maier. Logic and lattices for distributed programming. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 1. ACM, 2012.

[5] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order (2. ed.)*. Cambridge University Press, 2002.

[6] Bettina Kemme, André Schiper, G. Ramalingam, and Marc Shapiro. Dagstuhl seminar review: Consistency in distributed systems. *SIGACT News*, 45(1):67–89, March 2014.

[7] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Rapp. Rech. 7506, Institut National de la Recherche en Informatique et Automatique (INRIA), Rocquencourt, France, January 2011.

[8] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In Xavier Défago, Franck Petit, and V. Villain, editors, *Int. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS)*, volume 6976 of *Lecture Notes in Comp. Sc.*, pages 386–400, Grenoble, France, October 2011. Springer-Verlag.