Automated Monotone GC for Distributed Programming

Xinghao Pan
xinghao@eecs.berkeley.edu

Abstract

Edelweiss [1] provides automatic garbage collection for event log exhanges, programs which monotonically accumulate logs. However, in the process of doing so, it introduced additional points of coordination through non-monotone operations, thereby defeating the original purpose of ELEs to avoid synchronization and coordination. In this paper, we show that garbage collection for ELEs can in fact be monotone and coordination-free. We explicitly recast Edelweiss techniques as monotone operations on lattices defined over the input sets.

1 Approach

Plan of attack:

1. Promote all operations to tombstone sets.
2. Add (logical) GC rules.
   - Require that GC rules maintain a GC invariant, are monotone, and conservative (only promotes $\exists$ to $\top$ and not create new tuples)
   - Show that rewritten (logical) program is correct, monotone, and preserves GC invariant.
3. Instantiate representation, add instantiated GC (GCI) rules.
   - Require GCI rules delete only things that are (logically) tombstoned.
   - Show that representation maintains invariant – always keeps $\exists$ but possibly also tombstoned tuples.

Our approach proceeds in two phases. Firstly, we add (logical) rules that identifies 'tombstone' tuples – these are tuples whose removal does not affect the correctness of the program, in the sense that the output sets in the rewritten program are unaffected. We further require that the GC rules are monotone.

The second phase then rewrites the program to perform the actual deletion. Here, we ensure that deleted tuples are tombstoned, but tombtoned tuples are not necessarily deleted. This invariance ensures that the instantiated representation corresponds to the logical program in the first phase, and in particular, instantiated representations of the output sets (which have no tombstones) are correct.

However, we do not require the representation to be monotone; in particular, a deleted tombstone tuple could potentially be re-instantiated without affecting the correctness of the program. Nevertheless, since we are maintaining equivalence to the logical, monotone program, we may execute operations in a coordination-free manner.

1.1 Some thoughts

I think there are some connections to both CALM and $\mathcal{I}$-confluence.

1. The logical GC rules are monotone, so CALM tells us that the logical program is eventually consistent without any points of order.
2. The instantiated program is not monotone, so the representation is not eventually consistent without imposing points of order.
3. Throughout the paper, we introduce various invariants and demonstrate that they hold in the rewritten logical and instantiated programs (as long as the GC rules introduce fulfill some properties). In other words, we show $\mathcal{I}$-confluence of our rewritten rules and GC rules. The IC theorem tells us that we can have coordination-free execution and still "converge"[1]

[1] We need to be careful about what coordination-freedom, convergence, and transactional availability means in the context of a Bloom program.

2 Preliminaries

2.1 Tombstones

2.2 Copy rules — reduction for multi-output sets

2.3 Reduction for multi-input rules

Consider the rule `B <= f(A1, ..., Ak)`. Instead of working with k input sets, we can define a set `A` such that $(i, a) \in \texttt{A} \iff a \in \texttt{A}$. We may then replace the rule with `B <= f(A)`, and introduce rules `A <= map(`A_i`, i)`, where `map(S, j)` $= \{(j, s) : s \in \texttt{S}\}$. (Note that this rule can be garbage collected almost immediately, and should not incurred additional storage.)

Alternatively, we can modify the input rules to $\texttt{A}_i$'s. In conjunction with the copy rules, each $\texttt{A}_i$ should participate as input only to `f`. Hence, for each rule $\texttt{A}_i$ `<= g(...)`, we replace with `A <= map(g(...), i)`.

3 Logical rewrite

In this section, we present a logical rewrite of the original program. We first endow all sets with 'tombstone' sets – intuitively, tombstoned tuples are that which we may safely delete without affecting the outcome of our program. We then add logical GC rules that define how we can identify and mark tombstoned tuples. The logical GC rules have to obey certain properties which will ensure correctness and monotonicity.

In the remainder of this write-up, we will consider rewrite the general rule, for arbitrary `A`:

$$\texttt{B <= f(A)} \tag{3.1}$$

Note that we do not require `f` to be monotone. In cases where `f` is non-monotone, the program remains non-monotone; however, our rewrites do not introduce additional non-monotone operations, and thus there are no new points of order added. [2] For simplicity, we will assume that each input set only participates in one rule – we avoid multi-output situations by using copy rules, and handle copy rules separately.

3.1 Adding tombstones

Our first step is to lift all sets to a 3-phase set by endowing with an additional 'tombstone' state. Instead of having a single set `A`, we use a tuple $\texttt{A}_{\text{TS}} = (\texttt{A}_\neg, \texttt{A}_\exists, \texttt{A}_\top)$ of three mutually exclusive sets. Intuitively, tuples in $\texttt{A}_\top$ are those that would have been in the original set `A`, but are now also marked for reclamation. Conversely, tuples in $\texttt{A}_\exists$ are also in `A` but not marked for reclamation.

In cases where `A` has a unique key, we can additional keep a copy $\texttt{A}_!$ of the keys in the tuple $\texttt{A}_{TS!} = (\texttt{A}_\neg, \texttt{A}_\exists, \texttt{A}_\top, \texttt{A}_!)$ with the invariant $\texttt{A}_! = \pi_k(\texttt{A}_\exists \cup \texttt{A}_\top)$. The merge rule for such tuples is

$$\begin{aligned}\texttt{A}_{\text{TS!}} \sqcup \texttt{A}'_{\text{TS!}} &= (\texttt{A}_\neg, \texttt{A}_\exists, \texttt{A}_\top, \texttt{A}_!) \sqcup (\texttt{A}'_\neg, \texttt{A}'_\exists, \texttt{A}'_\top, \texttt{A}'_!) \\ &= (\texttt{A}_\neg \cup \texttt{A}'_\neg - (\texttt{A}_\exists \cup \texttt{A}'_\exists \cup \texttt{A}_\top \cup \texttt{A}_\top), \texttt{A}_\exists \cup \texttt{A}'_\exists - (\texttt{A}_\top \cup \texttt{A}_\top), \texttt{A}_\top \cup \texttt{A}_\top, \texttt{A}_! \cup \texttt{A}'_!)\end{aligned}$$

3.2 Logical garbage collection

Armed with this lifted representation, we can now provide a rewrite of the original program, together with an additional GC rule for each rule.

$$\text{Forward rule:} \quad \texttt{B}_{\text{TS}} \texttt{ <= } \exists\texttt{(f(A}_\exists \cup \texttt{A}_\top\texttt{))} \tag{3.2}$$

$$\text{GC rule:} \quad \texttt{A}_{\text{TS}} \texttt{ <= GC(A}_{\text{TS}}\texttt{, f, B}_{\text{TS}}\texttt{)} \tag{3.3}$$

The function $\exists(\texttt{S})$ takes a set `S` and returns $(\emptyset, \texttt{S}, \emptyset)$, and is a monotone function. The `GC` function takes TS sets and the functional relationship, and returns a new TS set with additional tombstoned tuples. We will make this notion formal below.

[2] I believe, however, that we will require that any set that appears as an input is monotonically growing, i.e. there are no deletions. This should be guaranteed by Edelweiss's requirement that there are no deletion rules.

For this rewrite to preserve correctness and monotonicity, we will require that the following invariance is maintained by the forward rule, the GC rule, and under arbitrary merges.

Invariant 3.1 (GC Invariance) *For any set* `A` *that participates in rules* `B_i <= f_i(A)` *for* $i = 1, \ldots, k$, *we require*

$$\forall i, \quad \forall \widehat{A} \supseteq A_\exists, \quad f_i(\widehat{A}) \cup B_{i,\exists} \cup B_{i,\top} = f_i(\widehat{A} \cup A_\top) \cup B_{i,\exists} \cup B_{i,\top} \tag{3.4}$$

Furthermore, for A_{TS} *and* A'_{TS} *satisfying Eq* (3.4), *it must be that*

$$\forall i, \quad \forall \widehat{A} \supseteq A_\exists \cup A'_\exists - (A_\top \cup A'_\top), \quad f_i(\widehat{A}) \cup B_{i,\exists} \cup B_{i,\top} = f_i(\widehat{A} \cup A_\top \cup A'_\top) \cup B_{i,\exists} \cup B_{i,\top} \tag{3.5}$$

Eq (3.4) ensures that any tombstoned tuple can be safely deleted, since any tuple it may generate (for any possible future input) is already present in the output set, and thus the tombstoned tuple has no downstream effect. The second condition Eq (3.5) is required to ensure that Eq (3.4) is preserved under merges for input sets $\mathtt{A}_{\mathrm{TS}}$.

Additionally, we require that the `GC` rule have the properties stated below.

Property 3.2 (GC Rule Invariance) *Any GC rule must maintain Invariant 3.1.*

Property 3.3 (GC Rule Monotonicity) *Suppose* $A_{TS} \leq A'_{TS}$, *and* $B_{TS} \leq B'_{TS}$. *If* (A_{TS}, B_{TS}) *and* (A'_{TS}, B'_{TS}) *both satisfy GC Invariant 3.1, then* `GC(A, f, B)` $\leq$ `GC(A', f, B')`.

Property 3.4 (GC Rule Conservation) *Let* $G_{TS} = (G_\neg, G_\exists, G_\top)$ = `GC(A, f, B)`. *Then it must be the case that* $G_\exists \subseteq A_\exists$, *and* $A_\top \subseteq G_\top \subseteq A_\exists \cup A_\top$, *and* $G_\exists \cup G_\top = A_\exists \cup A_\top$.

Property 3.3 ensures that the GC rules are monotone, and Property 3.4 only moves tuples from $\mathtt{A}_\exists$ to $\mathtt{A}_\top$.

3.2.1 Examples

While the above properties are reasonable expectations of garbage collection rules, it is not obvious that there are useful rules that satisfy them. We now show some examples of such rules.

Example 3.1 (Trivial GC) `GCtrivial(`A_{TS}`, f, `B_{TS}`)` $= A_{TS}$.

Example 3.2 (Copy GC) `GCcopy(`A_{TS}`, Id, `$B_{1,TS}$`, ..., , `$B_{k,TS}$`)` $= (\emptyset, A_\exists - B_\cap, A_\top \cup B_\cap)$, *where* $B_\cap = A_\exists \cap \bigcap_{i=1}^{k}(B_{i,\exists} \cup B_{i,\top})$.

For the copy rule, the GC Invariant 3.1 reduces to

$$\forall i, \mathtt{A}_\top \subseteq \mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top}, \tag{3.6}$$

$$\forall i, \mathtt{A}_\top \cup \mathtt{A}'_\top \subseteq \mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top}. \tag{3.7}$$

Suppose $\mathtt{A}_{\mathrm{TS}}$ satisfies the GC invariant Eq (3.6). Then, $\forall i, \mathtt{A}_\top \cup \mathtt{B}_\cap \subseteq \mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top}$, so `GCcopy` preserves Eq (3.6). Suppose $\mathtt{A}'_{\mathrm{TS}}$ satisfies the GC invariant Eq (3.6). Then $\forall i, \mathtt{A}_\top \cup \mathtt{B}_\cap \cup \mathtt{A}'_\top \subseteq \mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top}$, so `GCcopy` preserves Eq (3.7).

The GC Rule Conservation Property 3.4 is straightforward.

Suppose $\mathtt{A}'_{\mathrm{TS}} \geq \mathtt{A}_{\mathrm{TS}}$, i.e., $\mathtt{A}'_\top \supseteq \mathtt{A}_\top$ and $\mathtt{A}'_\top \cup \mathtt{A}'_\exists \supseteq \mathtt{A}_\top \cup \mathtt{A}_\exists$. Suppose also $\mathtt{A}'_{\mathrm{TS}}$ satisfies the GC Invariant 3.1 with respect to $\mathtt{B}'_{i,\mathrm{TS}} \geq \mathtt{B}_{i,\mathrm{TS}}$. Then

$$\begin{aligned}
\mathtt{A}_\top \cup \left(\mathtt{A}_\exists \cap \bigcap_{i=1}^{k}(\mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top})\right) &\subseteq \mathtt{A}'_\top \cup \left((\mathtt{A}'_\exists \cup \mathtt{A}'_\top) \cap \bigcap_{i=1}^{k}(\mathtt{B}'_{i,\exists} \cup \mathtt{B}'_{i,\top})\right) \\
&= \mathtt{A}'_\top \cup \left(\mathtt{A}'_\exists \cap \bigcap_{i=1}^{k}(\mathtt{B}'_{i,\exists} \cup \mathtt{B}'_{i,\top})\right) \cup \left(\mathtt{A}'_\top \cap \bigcap_{i=1}^{k}(\mathtt{B}'_{i,\exists} \cup \mathtt{B}'_{i,\top})\right) \\
&= \mathtt{A}'_\top \cup \left(\mathtt{A}'_\exists \cap \bigcap_{i=1}^{k}(\mathtt{B}'_{i,\exists} \cup \mathtt{B}'_{i,\top})\right),
\end{aligned}$$

where the final equality follows because $\forall i = 1, \ldots, k,\ \mathtt{A}'_\top \subseteq \mathtt{B}'_{i,\exists} \cup \mathtt{B}'_{i,\top}$. Furthermore, it follows from the GC Rule Conservation Property 3.4, we also have that $(\mathtt{A}_\exists - \mathtt{B}_\cap) \cup (\mathtt{A}_\top \cup \mathtt{B}_\cap) \subseteq (\mathtt{A}'_\exists - \mathtt{B}'_\cap) \cup (\mathtt{A}'_\top \cup \mathtt{B}'_\cap)$. Hence, `GCcopy` is monotone.

Example 3.3 (Max GC) *`GCmax(`A_{TS}`, f, `B_{TS}`)` $= (\emptyset, A_\exists - \widetilde{A}, \widetilde{A})$ where $A_\top \subseteq \widetilde{A} \subseteq A_\exists \cup A_\top$ is the (unique) largest set such that*

$$\forall \widehat{A} \supseteq A_\exists - \widetilde{A}, \quad f(\widehat{A}) \cup B_\exists \cup B_\top = f(\widehat{A} \cup \widetilde{A} \cup A_\top) \cup B_\exists \cup B_\top, \tag{3.8}$$

and for any A'_{TS} satisfying the GC Invariant 3.1 Eq (3.4)*,*

$$\forall \widehat{A} \supseteq A_\exists \cup A'_\exists - (\widetilde{A} \cup A_\top \cup A'_\top), \quad f(\widehat{A}) \cup B_\exists \cup B_\top = f(\widehat{A} \cup \widetilde{A} \cup A_\top \cup A'_\top) \cup B_\exists \cup B_\top. \tag{3.9}$$

The `GCmax` rule returns the largest set of tombstones that can be safely deleted without interfering with other tombstones. It also most directly attempts to maintain the GC invariant.

For this rule to make sense, we need to show that it is in fact well-defined, i.e., there is a unique largest set that achieves the conditions. Suppose there are sets $\mathtt{X}$ and $\mathtt{Y}$ that achieve both conditions. Then $\mathtt{Z} = \mathtt{X} \cup \mathtt{Y}$ also fulfills our two conditions. First, note that since $\mathtt{X}$ and $\mathtt{Y}$ both satisfy Eq (3.8) and Eq (3.9), the TS sets $(\mathtt{A}_\neg, \mathtt{A}_\exists - \mathtt{X}, \mathtt{X} \cup \mathtt{A}_\top)$ and $(\mathtt{A}_\neg, \mathtt{A}_\exists - \mathtt{Y}, \mathtt{Y} \cup \mathtt{A}_\top)$ satisfy Invariant 3.1. Applying Eq (3.9), we have

$$\begin{aligned} \forall \widehat{\mathtt{A}} \ &\supseteq\ \mathtt{A}_\exists \cup (\mathtt{A}_\exists - \mathtt{Y}) - (\mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{Y} \cup \mathtt{A}'_\top) \ &=\ \mathtt{A}_\exists - (\mathtt{X} \cup \mathtt{Y}) \ &=\ \mathtt{A}_\exists - \mathtt{Z}, \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{X} \cup \mathtt{A}_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{X} \cup \mathtt{Y} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top, \end{aligned}$$

where we have used the fact that $\widehat{\mathtt{A}} \cup \mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top \supseteq \mathtt{A}_\exists - \mathtt{Y}$ and Eq (3.8) for $\mathtt{Y}$. Hence, $\mathtt{Z}$ satisfies Eq (3.8).

Furthermore, for any $\mathtt{A}'_{\mathrm{TS}}$ satisfying the GC Invariant 3.1 Eq (3.4), applying Eq (3.9) to $\mathtt{X}$, we get

$$\forall \widehat{\mathtt{A}} \ \supseteq\ \mathtt{A}_\exists \cup \mathtt{A}'_\exists - (\mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top), \qquad \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \ =\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top,$$

so $\mathtt{A}''_{\mathrm{TS}} = (\emptyset, (\mathtt{A}_\exists \cup \mathtt{A}'_\exists) - (\mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top), \mathtt{X} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top)$ satisfies Eq (3.4). We can then apply Eq (3.9) to $\mathtt{Y}$:

$$\begin{aligned} \forall \widehat{\mathtt{A}} \ &\supseteq\ \mathtt{A}_\exists \cup \mathtt{A}''_\exists - (\mathtt{Y} \cup \mathtt{A}_\top \cup \mathtt{A}''_\top) \ &=\ \mathtt{A}_\exists \cup \mathtt{A}'_\exists - (\mathtt{X} \cup \mathtt{Y} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top), \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{Y} \cup \mathtt{A}_\top \cup \mathtt{A}''_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{X} \cup \mathtt{Y} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top. \end{aligned}$$

Hence $\mathtt{Z}$ also satisfies Eq (3.9). Thus, there is a unique largest set that achieves Eq (3.8) and (3.9).

It is easy to see `GCmax` maintains the GC Invariant 3.1, since Eq (3.8) satisfies Eq (3.4) and Eq (3.9) satisfies Eq (3.5). Similarly, the GC Rule Conservation Property 3.4 is also maintained by the choice of $\mathtt{A}_\top \subseteq \widetilde{\mathtt{A}} \subseteq \mathtt{A}_\exists \cup \mathtt{A}_\top$.

To show monotonicity, suppose we have $\mathtt{A}_{\mathrm{TS}} \leq \mathtt{A}'_{\mathrm{TS}}$ and $\mathtt{B}_{\mathrm{TS}} \leq \mathtt{B}'_{\mathrm{TS}}$. Monotonicity in the $\mathtt{B}$ argument is easy: Eq (3.8) and (3.9) for $\mathtt{B}_{\mathrm{TS}}$ immediately implies the same for $\mathtt{B}'_{\mathrm{TS}}$. For the $\mathtt{A}$ argument, we observe that $\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top$ satisfies conditions Eq (3.8) and Eq (3.9) as applied to $\mathtt{A}'_{\mathrm{TS}}$. For condition Eq (3.8),

$$\begin{aligned} \forall \widehat{\mathtt{A}} \supseteq \mathtt{A}'_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top) \ &=\ (\mathtt{A}'_\exists \cup (\mathtt{A}_\exists - \mathtt{A}'_\top)) - (\widetilde{\mathtt{A}} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top) \ &=\ \mathtt{A}'_\exists \cup \mathtt{A}_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}_\top \cup \mathtt{A}'_\top), \\ f(\widehat{\mathtt{A}}) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top \ &=\ f(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top, \end{aligned}$$

where we have applied Eq (3.9) for $\mathtt{A}_{\mathrm{TS}}$ and $\mathtt{B}'_{\mathrm{TS}}$. Also, for any $\mathtt{A}''_{\mathrm{TS}}$ that satisfies Eq (3.4),

$$\begin{aligned} \forall \widehat{\mathtt{A}} \ &\supseteq\ \mathtt{A}_\exists \cup \mathtt{A}''_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}_\top \cup \mathtt{A}''_\top) \ &=\ \mathtt{A}_\exists \cup \mathtt{A}''_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top), \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}_\top \cup \mathtt{A}''_\top) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top, \end{aligned}$$

so $(\emptyset, \mathtt{A}_\exists \cup \mathtt{A}''_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top), \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top)$ satisfies Eq (3.4). Repeating the process for $\mathtt{A}'_{\mathrm{TS}}$,

$$\begin{aligned} \forall \widehat{\mathtt{A}} \ &\supseteq\ \mathtt{A}'_\exists \cup (\mathtt{A}_\exists \cup \mathtt{A}''_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top)) - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top \cup \mathtt{A}''_\top) \ &=\ \mathtt{A}'_\exists \cup \mathtt{A}''_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top), \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{A}'_\top \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top \ &=\ \mathtt{f}(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \cup \mathtt{A}''_\top) \cup \mathtt{B}'_\exists \cup \mathtt{B}'_\top, \end{aligned}$$

satisfying Eq (3.9). Therefore, $\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top$ is a valid set of tombstones for `GCmax(`$\mathtt{A}_{\mathrm{TS}}$`', f, `$\mathtt{B}_{\mathrm{TS}}$`')`, and so `GCmax(`$\mathtt{A}_{\mathrm{TS}}$`', f, `$\mathtt{B}_{\mathrm{TS}}$`')` $\supseteq \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top \supseteq \widetilde{\mathtt{A}} =$ `GCmax(`$\mathtt{A}_{\mathrm{TS}}$`, f, `$\mathtt{B}_{\mathrm{TS}}$`)`.

We also point out that GCmax is complete in the sense that any $\widetilde{\mathtt{A}}$ that satisfies Eq (3.8) and (3.9) will necessarily be $\widetilde{\mathtt{A}} \subseteq \mathtt{GCmax}(\mathtt{A_{TS}}, \mathtt{f}, \mathtt{B_{TS}})$ due to the maximality of GCmax, so any tuple that could be deleted (per Eq (3.8) and (3.9)) will be tombstoned. (However, it may not be truly complete. For example, if we have $C <= A \bowtie B$, $D <= \pi_A(C)$, then we can always delete any tuple of B, but our rule does not allow for this.)

The GCmax rule may be hard to evaluate in practice, so we provide an easier rule below.

Example 3.4 (Tuple-based GC) $\mathtt{GC}(A_{TS}, \mathtt{f}, B_{TS}) = (\emptyset, A_\exists - \widetilde{A}, \widetilde{A})$ *where* $\widetilde{A} = A_\top \cup \{t \in A_\exists : \forall X, \mathtt{f}(X \cup \{t\}) - \mathtt{f}(X) \subseteq B_\exists \cup B_\top\}$.

3.3 Invariance, Correctness, Monotonicity

Theorem 3.1 (GC Invariance) *The GC Invariant 3.1 is maintained by the rewritten program.*

Proof: Our proof proceeds by showing that the initial conditions satisfy the invariant, and the forward, GC rules and merges preserve the invariant.

The program is initialized with $\mathtt{A}_\exists = \mathtt{A}$ and $\mathtt{A}_\top = \emptyset$, so the invariant is immediately satisfied.

There are two forward rules that we consider: $\mathtt{B_{TS}}$ <= $\exists(\mathtt{f}(\mathtt{A}_\exists \cup \mathtt{A}_\top))$ as well as rules that merge into $\mathtt{A_{TS}}$. In the first case, we increase $\mathtt{B}_\exists$ while keeping $\mathtt{B}_\top$ constant, so Eq (3.4) and (3.5) are preserved. In the second case, $\mathtt{A}_\exists$ is increased while $\mathtt{A}_\top$ is kept constant – let $\mathtt{A}_\exists^{t+1}$ be the new value of $\mathtt{A}_\exists$. Since $\mathtt{A}_\exists^{t+1} \supseteq \mathtt{A}_\exists$, the GC invariance of $\mathtt{A_{TS}}$ implies the GC invariance of $\mathtt{A}_{\mathrm{TS}}^{t+1}$.

GC on $\mathtt{A_{TS}}$ maintains the GC invariance due to the GC Rule Invariance Property 3.2. GC on $\mathtt{B_{TS}}$ maintains the GC invariance due to the GC Rule Conservation Property 3.4.

Merges on $\mathtt{B_{TS}}$ increases $\mathtt{B}_\exists \cup \mathtt{B}_\top$, so the GC invariance is immediately preserved.

Finally, we consider a merge of $\mathtt{A_{TS}}$ and $\mathtt{A}'_{\mathrm{TS}}$. Due to Eq (3.9), the merge of $\mathtt{A_{TS}}$ and $\mathtt{A}'_{\mathrm{TS}}$ preserves Eq (3.4). Now if we have a further $\mathtt{A}''_{\mathrm{TS}}$ satisfying Eq (3.4), we note that (1) Eq (3.5) for $\mathtt{A_{TS}}$ implies that the merge of $\mathtt{A_{TS}}$ and $\mathtt{A}''_{\mathrm{TS}}$ preserves Eq (3.4), and thus (2) Eq (3.5) for $\mathtt{A}''_{\mathrm{TS}}$ implies that the merge of $\mathtt{A}''_{\mathrm{TS}}$ with the merge of $\mathtt{A_{TS}}$ and $\mathtt{A}''_{\mathrm{TS}}$ preserves Eq (3.4). Together, this implies that Eq (3.5) is maintained through a merge of $\mathtt{A_{TS}}$ with $\mathtt{A}'_{\mathrm{TS}}$. □

Theorem 3.2 (Monotonicity) *The program rewrite does not introduce new points of order.*

Proof: Monotonicity of GC rules is implied via GC Rule Monotonicity 3.3. Furthermore, if f is monotone, then the composition of $\exists \circ \mathtt{f}$ is also monotone, so the forward rule is monotone. □

To show correctness of the logical program rewrite, we will show that the following invariant is maintained.

Invariant 3.5 (Logical Invariant) $A = A_\exists \cup A_\top$.

Theorem 3.3 (Correctness) *The program maintains the Logical Invariant 3.5. In particular, for output sets* A *which have no downstream operations, we have* $A_\top = \emptyset$ *so* $A = A_\exists$.

Proof: We will proceed via induction on the execution of the rewritten program. Let $\mathtt{A}_{\mathrm{TS}}^t$ denote the value of $\mathtt{A_{TS}}$ after executing t rules in the rewritten program. Simultaneously, we run the original program, executing the corresponding step whenever the forward rule or a merge is executed, and performing a noop when we run a GC rule. (Note that every execution of the original program corresponds to some execution of the rewritten program.) Using analogous notation, let $\mathtt{A}^t$ denote the value of A after executing t rules of the rewritten program (so $\mathtt{A}^{t+1} = \mathtt{A}^t$ if we run a GC rule).

Observe that the initialization gives $\mathtt{A}^0 = \mathtt{A}_\exists^0$.

Consider the execution of the forward rule $\mathtt{B_{TS}}$ <= $\exists(\mathtt{f}(\mathtt{A}_\exists \cup \mathtt{A}_\top)$, which we can equivalently write as $\mathtt{B}_\exists^t = \mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1})$, and $\mathtt{B}_\top^t = \mathtt{B}_\top^{t-1}$. Thus,

$$\begin{aligned}
\mathtt{B}_\exists^t \cup \mathtt{B}_\top^t &= \mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1}) \cup \mathtt{B}_\top^t \\
&= \mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}^{t-1}) \cup \mathtt{B}_\top^{t-1} \\
&= \mathtt{B}^{t-1} \cup \mathtt{f}(\mathtt{A}^{t-1}) \\
&= \mathtt{B}^t.
\end{aligned}$$

No other sets are altered by the forward rule.

The GC Rule Conservation Property 3.4 ensures that $\mathtt{A}^t_\exists \cup \mathtt{A}^t_\top = \mathtt{A}^{t-1}_\exists \cup \mathtt{A}^{t-1}_\top = \mathtt{A}^{t-1} = \mathtt{A}^t$.

Finally, consider the merge of $\mathtt{A}^{t-1}_{\mathrm{TS}}$ with $\mathtt{A}'^{(t-1)}_{\mathrm{TS}}$:

$$\mathtt{A}^t_\exists \cup \mathtt{A}^t_\top \;=\; \mathtt{A}^{t-1}_\exists \cup \mathtt{A}^{t-1}_\top \cup \mathtt{A}'^{(t-1)}_\exists \cup \mathtt{A}'^{(t-1)}_\top \;=\; \mathtt{A}^{t-1} \cup \mathtt{A}'^{(t-1)} \;=\; \mathtt{A}^t$$

Thus, we have $\mathtt{A} = \mathtt{A}_\exists \cup \mathtt{A}_\top$. Furthermore, since output sets have no downstream operations, they have no associated GC rules, and never gather any tombstones. Therefore for an output set A, we always maintain $\mathtt{A} = \mathtt{A}_\exists \cup \mathtt{A}_\top = \mathtt{A}_\exists$. □

4 Representation

In the previous section, we described a rewrite with logical GC which ensures correctness and monotonicity. However, the garbage collection only marks tuples for possible deletion, and thus does not actually reclaim any storage. Here, we present a second rewrite that does in fact reclaim storage.

To achieve storage reclamation, our second rewrite will use a representation that is non-monotone. However, the representation maintains an invariance with respect to the first program rewrite, so that an execution of the second rewrite corresponds to some execution of the first. As a result, monotonicity is preserved at the level of the logical program. In practice, this allows us to perform GC in a monotone, coordination-free manner.

4.1 Representation and rewrite

Instead of maintaining two sets $\mathtt{A}_\exists$ and $\mathtt{A}_\top$, we only instantiate a set $\mathtt{A}_I$ which holds all of $\mathtt{A}_\exists$ but possibly some tuples from $\mathtt{A}_\top$. Intuitively, $\mathtt{A}_I$ deletes a subset of the tombstoned tuples, which from the previous section, we know to be safe for deletion.

In place of the forward rule Eq (3.2) and GC rule Eq (3.3), we have *instantiated* rules that operate only on the instantiated sets:

$$\text{Instantiated forward rule:} \qquad \mathtt{B}_I \texttt{ <= } \mathtt{f}(\mathtt{A}_I) \tag{4.1}$$

$$\text{Instantiated GC rule:} \qquad \mathtt{A}_I \texttt{ <\# GCI(}\mathtt{A}_I\texttt{, f, }\mathtt{B}_I\texttt{)} \tag{4.2}$$

Here, we use `<#` to represent a non-deferred deletion.

In some cases, where `A` has a primary key, we may choose to augment the representation by keeping the primary keys of $\mathtt{A}_\exists \cup \mathtt{A}_\top$. We represent this as `A.k` in range compressed form, and write $\mathtt{A}_+ = (\mathtt{A.k}, \mathtt{A}_I)$ as the augmented representation. The merge of augmented representations is defined as $(\mathtt{A.k}, \mathtt{A}_I) \sqcup (\mathtt{A'.k}, \mathtt{A}'_I) = (\mathtt{A.k} \cup \mathtt{A'.k}, (\mathtt{A}_I \cup \mathtt{A}'_I)\texttt{.notin(A.k.notin(}\mathtt{A}_I\texttt{))} \cup \mathtt{A'.k.notin}(\mathtt{A}'_I))$.

The rules Eq (4.1) and (4.2) for augmented representation are:

$$\text{Instantiated (augmented) forward rule:} \qquad \mathtt{B}_+ \texttt{ <= } (\pi_k(\mathtt{f}(\mathtt{A}_I)), \mathtt{f}(\mathtt{A}_I)) \tag{4.3}$$

$$\text{Instantiated (augmented) GC rule:} \qquad \mathtt{A}_+ \texttt{ <= } (\mathtt{A.k}, \mathtt{GCI}(\mathtt{A}_+\texttt{, f, }\mathtt{B}_+\texttt{)}) \tag{4.4}$$

We require that the instantiated GC rule to have the following property, which states that `GCI` only deletes tuples that the corresponding `GC` rule has marked as tombstoned.

Property 4.1 (Instantiated GC Safe Deletion) *If $A_\exists \subseteq A_I \subseteq A_\exists \cup A_\top$ and $B_\exists \subseteq B_I \subseteq B_\exists \cup B_\top$, then `GCI(`$A_I$`, f, `$B_I$`)` $\subseteq \widetilde{A}$ where $(\emptyset, \widehat{A}, \widetilde{A})$ = `GC(`A_{TS}`, f, `B_{TS}`)`. Furthermore, if `A.k` $= \pi_k(A_\exists \cup A_\top)$, then `GCI(`$A_+$`, f, `$B_+$`).k` $=$ `GC(`A_{TS}`, f, `B_{TS}`).k`.*

4.2 Correctness

The correctness of this program rewrite is demonstrated by maintaining the following invariant.

Invariant 4.2 (Representation Invariant) $A_\exists \subseteq A_I \subseteq A_\exists \cup A_\top$. *For augmented representations, `A.k` $= \pi_k(A_\exists \cup A_\top)$.*

The Representation Invariant 4.2 states that the instantiated set does not contain superfluous tuples, and it does not delete tuples that are unsafe for deletion. Furthermore, for the augmented representation, we can retrieve the primary keys of tombstoned tuples by $\pi_k(\mathtt{A}_\top) = \mathtt{A.k.notin}(\mathtt{A}_I, \mathtt{:k => :k})$.

Theorem 4.1 (Representation Correctness) *The Representation Invariant 4.2 is maintained by the instantiated program. In particular, for output sets $\mathtt{A}$ which have no downstream operations, we have $\mathtt{A} = \mathtt{A}_I$.*

Proof: We initialize the program with $\mathtt{A}_I^0 = \mathtt{A}^0 = \mathtt{A}_\exists^0 = \mathtt{A}_\exists^0 \cup \mathtt{A}_\top^0$ — the final two equalities are due to Thm 3.3.

First, consider the instantiated forward rule $\mathtt{B}_I$ `<= f(`$\mathtt{A}_I$`)`, which only alters $\mathtt{B}_I$ and can be expressed as $\mathtt{B}_I^t = \mathtt{B}_I^{t-1} \cup \mathtt{f}(\mathtt{A}_I^{t-1})$.

$$
\begin{aligned}
\mathtt{B}_\exists^t &= (\mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1})) - \mathtt{B}_\top^{t-1} \\
&= (\mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1}) \cup \mathtt{B}_\top^{t-1}) - \mathtt{B}_\top^{t-1} \\
&= \mathtt{B}_\exists^{t-1} \cup \mathtt{f}(\mathtt{A}_I^{t-1}) - \mathtt{B}_\top^{t-1} && \text{(Thm 3.1)} \\
&\subseteq \mathtt{B}_I^{t-1} \cup \mathtt{f}(\mathtt{A}_I^{t-1}) && = \mathtt{B}_I^t \\
&\subseteq \mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1} \cup \mathtt{f}(\mathtt{A}_I^{t-1}) \\
&= \mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1} \cup \mathtt{f}(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1}) && \text{(Thm 3.1)} \\
&= \mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1}
\end{aligned}
$$

For the augmented forward rule, we have

$$
\begin{aligned}
\mathtt{B}^t\mathtt{.k} &= \mathtt{B}^{t-1}\mathtt{.k} \cup \mathtt{f}(\mathtt{A}_I^{t-1})\mathtt{.k} \\
&= \pi_k(\mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1}) \cup \pi_k(f(\mathtt{A}_I^{t-1})) \\
&= \pi_k(\mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1} \cup f(\mathtt{A}_I^{t-1})) \\
&= \pi_k(\mathtt{B}_\exists^{t-1} \cup \mathtt{B}_\top^{t-1} \cup f(\mathtt{A}_\exists^{t-1}\mathtt{A}_\top^{t-1})) \\
&= \pi_k(\mathtt{B}_\exists^t \cup \mathtt{B}_\top^t).
\end{aligned}
$$

Next we show that the instantiated GC rule maintains the invariant.

$$
\begin{aligned}
\mathtt{A}_\exists^t &= \mathtt{A}_\exists^{t-1} - \widetilde{\mathtt{A}} \\
&\subseteq \mathtt{A}_I^{t-1} - \mathtt{GCI}(\mathtt{A}_I\mathtt{,\ f,\ B}_I) && \text{(Property 4.1)} \\
&= \mathtt{A}_I^t \\
&\subseteq \mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1} \\
&= \mathtt{A}_\exists^t \cup \mathtt{A}_\top^t && \text{(Property 3.4)}
\end{aligned}
$$

For the augmented GC rule, we have $\mathtt{A}^t\mathtt{.k} = \mathtt{A}^{t-1}\mathtt{.k} = \pi_k(\mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1}) = \pi_k(\mathtt{A}_\exists^t \cup \mathtt{A}_\top^t)$.

Lastly, we consider the merge $\mathtt{A}_I^t = \mathtt{A}_I^{t-1} \cup \mathtt{A}_I'^{(t-1)}$ with the corresponding merge $\mathtt{A}_{\mathrm{TS}}^t = \mathtt{A}_{\mathrm{TS}}^{t-1} \sqcup \mathtt{A}_{\mathrm{TS}}'^{(t-1)}$.

$$
\begin{aligned}
\mathtt{A}_\exists^t &= \mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\exists'^{(t-1)} - (\mathtt{A}_\top^{t-1} \cup \mathtt{A}_\top'^{(t-1)}) \\
&= \mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\exists'^{(t-1)} \\
&\subseteq \mathtt{A}_I^{t-1} \cup \mathtt{A}_I'^{(t-1)} && = \mathtt{A}_I^t \\
&\subseteq \mathtt{A}_\exists^{t-1} \cup \mathtt{A}_\top^{t-1} \cup \mathtt{A}_\exists'^{(t-1)} \cup \mathtt{A}_\top'^{(t-1)} \\
&= \mathtt{A}_\exists^t \cup \mathtt{A}_\top^t.
\end{aligned}
$$

Hence, we have shown that the Representation Invariant 4.2 is maintained by the instantiated program. Furthermore, Thm 3.3 tells us that for output sets with no downstream operations, we have $\mathtt{A}_\top = \emptyset$, so $\mathtt{A}_I = \mathtt{A}_\exists = \mathtt{A}$. □

4.3 Examples

Example 4.1 (Instantiated Trivial GC) $GCIcopy(A_I, f, B_I) = \emptyset$.

Example 4.2 (Instantiated Copy GC) $GCIcopy(A_I, Id, B_{1,I}, \ldots, B_{k,I}) = A_I \cap \bigcap_{i=1}^{k} B_{i,I}$.

The GCcopy rule creates tombstones for $\mathtt{A}_\top \cup \left(\mathtt{A}_\exists \cap \bigcap_{i=1}^{k}(\mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top})\right)$. From the Representation Invariant 4.2, we know $\mathtt{A}_\exists \subseteq \mathtt{A}_I \subseteq \mathtt{A}_\exists \cup \mathtt{A}_\top$ and $\mathtt{B}_\exists \subseteq \mathtt{B}_I \subseteq \mathtt{B}_\exists \cup \mathtt{B}_\top$. Thus,

$$\mathtt{A}_I \cap \bigcap_{i=1}^{k} \mathtt{B}_{i,I} \;\subseteq\; (\mathtt{A} \cup \mathtt{A}_\top) \cap \bigcap_{i=1}^{k}(\mathtt{B}_\exists \cup \mathtt{B}_\top) \;\subseteq\; (\mathtt{A} \cup \mathtt{A}_\top) \cap \left(\mathtt{A}_\top \cup \bigcap_{i=1}^{k}(\mathtt{B}_\exists \cup \mathtt{B}_\top)\right) \;=\; \mathtt{A}_\top \cup \left(\mathtt{A}_\exists \cap \bigcap_{i=1}^{k}(\mathtt{B}_{i,\exists} \cup \mathtt{B}_{i,\top})\right),$$

so GCIcopy satisfies the Instantiated GC Safe Deletion Property 4.1.

Example 4.3 (Instantiated Max GC) $GCImax(A_I, f, B_I) = \widetilde{A}$, *where* $\widetilde{A} \subseteq A_I$ *is a set satisfying*

$$\forall \widehat{A} \supseteq A_I - \widetilde{A}, \quad f(\widehat{A}) \cup B_I = f(\widehat{A} \cup \widetilde{A}) \cup B_I, \tag{4.5}$$

and for all A'_{TS} *satisfying the GC Invariant 3.1 Eq 3.4,*

$$\forall \widehat{A} \supseteq A_I \cup A'_\exists - (\widetilde{A} \cup A'_\top), \quad f(\widehat{A}) \cup B_I = f(\widehat{A} \cup \widetilde{A} \cup A'_\top) \cup B_I \tag{4.6}$$

We first point out that the GCImax rule does not require knowledge of the value of any actual $\mathtt{A}'_{\mathrm{TS}}$, as it has to hold true for *all* such sets.

Next, we show that $\widetilde{\mathtt{A}} \cup \mathtt{A}_\top$ satisfies conditions Eq (3.8) and (3.9). The first condition Eq (4.5), together with the fact that $\mathtt{B}_I \subseteq \mathtt{B}_\top \cup \mathtt{B}_\exists$ implies that

$$\forall \widehat{\mathtt{A}} \supseteq \mathtt{A}_I - \widetilde{\mathtt{A}}, \quad \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top = \mathtt{f}(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top,$$

so $(\emptyset, \mathtt{A}_I - \widetilde{\mathtt{A}}, \widetilde{\mathtt{A}})$ satisfy GC Invariant 3.1 Eq 3.4. Applying Eq 3.5, we get

$$\begin{aligned} \forall \widehat{\mathtt{A}} \;&\supseteq\; \mathtt{A}_\exists \cup (\mathtt{A}_I - \widetilde{\mathtt{A}}) - (\mathtt{A}_\top \cup \widetilde{\mathtt{A}}) \;=\; \mathtt{A}_\exists - (\mathtt{A}_\top \cup \widetilde{\mathtt{A}}), \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \;&=\; \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{A}_\top \cup \widetilde{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top, \end{aligned}$$

thus satisfying Eq (3.8) with tombstones $\widetilde{\mathtt{A}} \cup \mathtt{A}_\top$. Similarly, Eq (4.3) implies

$$\forall \widehat{\mathtt{A}} \supseteq \mathtt{A}_I \cup \mathtt{A}'_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top), \quad \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top = \mathtt{f}(\widehat{\mathtt{A}} \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top, \tag{4.7}$$

so $(\emptyset, \mathtt{A}_I \cup \mathtt{A}'_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top), \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top)$ satisfy GC Invariant 3.1 Eq 3.4. Applying Eq 3.5, we get

$$\begin{aligned} \forall \widehat{\mathtt{A}} \;&\supseteq\; \mathtt{A}_\exists \cup (\mathtt{A}_I \cup \mathtt{A}'_\exists - (\widetilde{\mathtt{A}} \cup \mathtt{A}'_\top)) - (\mathtt{A}_\top \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top) \;=\; \mathtt{A}_\exists \cup \mathtt{A}'_\exists - (\mathtt{A}_\top \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top), \\ \mathtt{f}(\widehat{\mathtt{A}}) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top &= \mathtt{f}(\widehat{\mathtt{A}} \cup \mathtt{A}_\top \cup \widetilde{\mathtt{A}} \cup \mathtt{A}'_\top) \cup \mathtt{B}_\exists \cup \mathtt{B}_\top \end{aligned}$$

thus satisfying Eq (3.9) with tombstones $\widetilde{\mathtt{A}} \cup \mathtt{A}_\top$. We have therefore shown that $\mathtt{GCImax}(\mathtt{A}_I, \mathtt{f}, \mathtt{B}_I) = \widetilde{\mathtt{A}} \subseteq \widetilde{\mathtt{A}} \cup \mathtt{A}_\top \subseteq \mathtt{GCmax}(\mathtt{A}_{\mathtt{TS}}, \mathtt{f}, \mathtt{B}_{\mathtt{TS}})$ due to the maximality of GCmax.

Example 4.4 (Instantiated Tuple-based GC) $GCItuple(A_I, f, B_I) = \{t \in A_I \;:\; \forall X, f(X \cup \{t\}) - f(X) \subseteq B_I\}$

4.4 Putting it together

Our instantiated representation preserves correctness, and allows for GC opportunities. While it is not monotone in the representation, it tracks the monotone logical program, and thus our GC operations can be performed coordination-free.

5 Concrete Rules

5.1 DR+

The DR+ technique reclaims storage whenever a set difference is observed in the Bloom program. Specifically, we identify patterns of the form

$$\texttt{Z <= X.notin(Y, :kx => :ky)}$$

Letting $\widetilde{\texttt{X}} = ((\texttt{X}_\exists \cup \texttt{X}_\top) * (\texttt{Y}_\exists \cup \texttt{Y}_\top))\texttt{.lefts(:kx => :ky)}$, the logical reclamation rule is

$$\texttt{Z}_{\text{TS}} \texttt{ <= } (\texttt{X}_\exists \cup \texttt{X}_\top)\texttt{.notin}(\texttt{Y}_\exists \cup \texttt{Y}_\top, \texttt{:kx => :ky}) \tag{5.1}$$

$$\texttt{X}_{\text{TS}} \texttt{ <= } (\emptyset, \texttt{X}_\exists - \widetilde{\texttt{X}}, \texttt{X}_\top \cup \widetilde{\texttt{X}}) \tag{5.2}$$

and the instantiated reclamation rule is

$$\texttt{Z}_I \texttt{ <= X}_I\texttt{.notin}(\texttt{Y}_I, \texttt{:kx => :ky}) \tag{5.3}$$

$$\texttt{X}_I \texttt{ <\# } (\texttt{X}_I * \texttt{Y}_I)\texttt{.lefts(:kx -> :ky).} \tag{5.4}$$

It is easy to show that the logical rule Eq (5.2) is monotone and conservative. We also claim that the GC Invariant 3.1 Eq (3.4) reduces to

$$\texttt{X}_\top\texttt{.notin}(\texttt{Y}_\exists \cup \texttt{Y}_\top, \texttt{:kx => :ky}) = \emptyset,$$

and GC Invariant 3.1 Eq (3.5) reduces to if $(\texttt{X}_{\text{TS}}, \texttt{Y}_{\text{TS}})$ and $(\texttt{X}'_{\text{TS}}, \texttt{Y}'_{\text{TS}})$ satisfy Eq (4.5), then

$$(\texttt{X}_\top \cup \texttt{X}'_\top)\texttt{.notin}(\texttt{Y}_\exists \cup \texttt{Y}_\top \cup \texttt{Y}'_\exists \cup \texttt{Y}'_\top, \texttt{:kx => :ky}) = \emptyset.$$

We now show that the logical `GC` rule satisfies Eq (3.8) and Eq (3.9):

$$\begin{aligned}
&\texttt{X}^t_\top\texttt{.notin}(\texttt{Y}^t_\exists \cup \texttt{Y}^t_\top, \texttt{:kx => :ky}) \\
&= (\texttt{X}^{t-1}_\top \cup \widetilde{\texttt{X}})\texttt{.notin}(\texttt{Y}^{t-1}_\exists \cup \texttt{Y}^{t-1}_\top, \texttt{:kx => :ky}) \\
&= \widetilde{\texttt{X}}\texttt{.notin}(\texttt{Y}^{t-1}_\exists \cup \texttt{Y}^{t-1}_\top, \texttt{:kx => :ky}) \\
&= ((\texttt{X}^{t-1}_\exists \cup \texttt{X}^{t-1}_\top) * (\texttt{Y}^{t-1}_\exists \cup \texttt{Y}^{t-1]}_\top))\texttt{.lefts(:kx => :ky).notin}(\texttt{Y}^{t-1}_\exists \cup \texttt{Y}^{t-1}_\top, \texttt{:kx => :ky}) \\
&= \emptyset,
\end{aligned}$$

$$\begin{aligned}
&(\texttt{X}^t_\top \cup \texttt{X}'^t_\top)\texttt{.notin}(\texttt{Y}^t_\exists \cup \texttt{Y}^t_\top \cup \texttt{Y}'^t_\exists \cup \texttt{Y}'^t_\top, \texttt{:kx => :ky}) \\
&\subseteq \texttt{X}^t_\top\texttt{.notin}(\texttt{Y}^t_\exists \cup \texttt{Y}^t_\top, \texttt{:kx => :ky}) \quad \cup \quad \texttt{X}'^t_\top\texttt{.notin}(\texttt{Y}'^t_\exists \cup \texttt{Y}'^t_\top, \texttt{:kx => :ky}) \\
&= \emptyset.
\end{aligned}$$

Furthermore, we can show that the DR+ rule is a max GC rule. For, AFSOC that some $\widetilde{\texttt{W}} \not\subseteq \widetilde{\texttt{X}}$, then there is some $w \in \widetilde{\texttt{W}} \subseteq (\texttt{X}_\exists \cup \texttt{X}_\top)$ such that $w \notin \widetilde{\texttt{X}}$ but also $\{w\}\texttt{.notin}(\texttt{Y}_\exists \cup \texttt{Y}_\top, \texttt{:kx => :ky}) = \emptyset$. But that would imply $w \in ((\texttt{X}_\exists \cup \texttt{X}_\top) * (\texttt{Y}_\exists \cup \texttt{Y}_\top))\texttt{.lefts(:kx => :ky)} = \widetilde{\texttt{X}}$.

It remains to show that the instantiated GC rule satisfies the Instantiated GC Safe Deletion Property 4.1, which follows straightforwardly from the fact that $\texttt{X}_I \subseteq \texttt{X}_\exists \cup \texttt{X}_\top$ and $\texttt{Y}_I \subseteq \texttt{Y}_\exists \cup \texttt{Y}_\top$:

$$(\texttt{X}_I * \texttt{Y}_I)\texttt{.lefts(:kx => :ky)} \subseteq ((\texttt{X}_\exists \cup \texttt{X}_\top) * (\texttt{Y}_\exists \cup \texttt{Y}_\top))\texttt{.lefts(:kx => :ky)}$$

Observe that we do not have to prove that the output of `Z` is consistent with the original program — this is guaranteed by our invariants and Representation Correctness Thm 4.1.

5.2 Joins with punctuations

$$\{x \in \mathtt{X}_\exists \cup \mathtt{X}_\top \ : \ \exists p \in \mathtt{Py} \text{ s.t. } (p.k = x.k) \wedge ((x * p.y) \in (\mathtt{Z}_\exists \cup \mathtt{Z}_\top))\}$$
$$\{y \in \mathtt{Y}_\exists \cup \mathtt{Y}_\top \ : \ \exists p \in \mathtt{Px} \text{ s.t. } (p.k = y.k) \wedge ((y * p.x) \in (\mathtt{Z}_\exists \cup \mathtt{Z}_\top))\}$$

$$\mathtt{GCX}(\mathtt{X}_I, \mathtt{Y}_I, \mathtt{Px}, \mathtt{Py}, \mathtt{join}, \mathtt{Z}_I) = \{x \in \mathtt{X}_I \ : \ \exists p \in \mathtt{Py} \text{ s.t. } (p.k = x.k) \wedge ((x * p.y) \in \mathtt{Z}_I)\}$$
$$\mathtt{GCY}(\mathtt{X}_I, \mathtt{Y}_I, \mathtt{Px}, \mathtt{Py}, \mathtt{join}, \mathtt{Z}_I) = \{y \in \mathtt{Y}_I \ : \ \exists p \in \mathtt{Px} \text{ s.t. } (p.k = y.k) \wedge ((y * p.x) \in \mathtt{Z}_I)\}$$

$$\mathtt{GC}(\mathtt{X}_I, \mathtt{Y}_I, \mathtt{Px}, \mathtt{Py}, \mathtt{join}, \mathtt{Z}_I)$$

References

[1] N. Conway, P. Alvaro, E. Andrews, and J. M. Hellerstein. Edelweiss: Automatic storage reclamation for distributed programming. *Proceedings of the VLDB Endowment*, 7(6):481–492, 2014.