# On the Definition of Operation-based Merging

## 1 Informal Abstract/Intro Outline

### 1.1 Background/Motivation

Existing definitions suffer from one or more of the following problems[1]:

**A)** They are implicit. Most of the time they are "definition by solution" of the form: "here is my algorithm that has postcondition $X$ (often not stated at all), therefore $X$ is the definition of the (merging) problem".

**B)** They are unrealizable (algorithmically), such as Sun et al. [1998, def. 3 and 4.3][2]: "the *intention* of each operation must be preserved at every replica". Equivalent to "the computer(s) must read my mind".

**C)** They lead to one or both of the following paradoxes:

**(C.1)** The (general) paradox of operation-based merging: more information (about intermediate operations/states) results in more, not fewer, conflicts. It is implied by those definitions that require commutations of the exact operations involved.

**(C.2)** The paradox of lost *future* updates. A (past) operation guarantees that a certain subset of future operations will be lost upon a merge[3]. Concrete example: `darcs`.

The *weave* concept of Lippe and van Oosterom [1992] suffers from both paradoxes.

**D)** They are intricately tied to a particular setting, making generalization not immediate. We will invoke J. D. H. Smith's often cited quote here: "What looks messy and complicated in a particular framework may turn out to be simple and obvious in the proper general one" [Smith, 1976, p. iv].

A number of optimistic replication systems attempt to provide a generic framework for merging. Some systems allow the user to formulate a merging algorithm (for a particular type of input) in a dedicated language [Terry et al., 1995]. Other systems allow the user to state some postconditions for the merge, which are then used as solution selectors for a generic algorithm that searches the state space [Kermarrec et al., 2001].

Debating the merits of these types of frameworks goes beyond the scope of this paper. Instead, we note that the problem addressed in this paper is orthogonal[4] to the existence of such frameworks. Before the user can hope to formulate either an algorithm or even a postcondition for a merging problem (on a particular type of input), the user must have a notion (definition) of what merging actually means.

### 1.2 Contributions

Since a definition by definition (sic) cannot be proved, we hope to convince that our definition of operation-based merging should be accepted by showing that it avoids the shortcomings previously enumerated.

We formulate our definition in the general setting of a transformation monoid[5], also known as an $M$-act, which we subclass to model conflicts — the *merge monoid*. In order appearance, the strengths of our definition are:

1. We avoid the (general) paradox of operation-based merging by giving the definition of a conflict-free merge in terms of *safe embeddings* that commute. The necessity of the safe embedding notion has already been observed by Ramsey and Csirmaz [2001] in the context of filesystem merging. They use the rather unfortunate term "approximation" for it. We merely generalize their notion.

2. We avoid the paradox of lost future updates by making the following key observations about unification — the elimination of duplicate operations:

   (i) Unification is an automated *resolution* mechanism for *false-conflicts*. It is *not* applicable if there exists a conflict-free merge.

   (ii) In turn, false-conflicts can only be defined in terms of *zeropotent* operations. Roughly speaking, a zeropotent operation gives "conflict everywhere" when composed with itself.

Examples and counterexamples for the above, as well as their precise formulation, are provided in section 2.4.

3. Rather surprisingly, a useful definition of dependency can be given without making any additional assumption about the nature of the states. In our framework we are able to prove the intuitive truths that dependencies preclude commutation and vice versa.

Our notion of dependency is more discriminating than the event-based happened-before relation of Lamport [1978], in the sense that an operation $\beta$ may happen after another operation $\alpha$ everywhere, but this need not imply that $\beta$ depends

[1] I realize that some if not all of this stuff is very "bare knuckles", but nevertheless true. I need to "preen" it so it won't sound so shocking (w.r.t. what gets published as research in some reputable places).

[2] Not an exact quote. Their original definition is far more convoluted, but means exactly this. Also, they assume that an observed ordering at a replica implies a dependency (def. 1 in their paper), which is simply false.

[3] We take the liberty of using "merge" as a noun, synonym with reconciliation. This is a common practice in the revision control community.

[4] Frankly, it is a precondition for their usefulness. But, I want this paper to get published at PODC.

[5] For the reader unfamiliar with abstract algebra, we introduce the notions used in this paper at the beginning of the next section.

on $\alpha$. Although this distinction may be plain obvious to the reader, some previous research on merging applied Lamport's celebrated result inappropriately [Sun et al., 1998, def. 1].

4. We introduce the δ-act algebraic structure, which captures the basic behavior of the text merging programs `patch` and `diff`.

We also introduce an algebraic three-way merge operator that needs to satisfy two simple axioms[6], which can be trivially verified to hold for important algorithms. E.g., they can be verified for the widely-used[7] `diff3` text merging algorithm based on the excellent analysis due to Khanna, Kunal, and Pierce [2007].

We show that a three-way merge operator can be used to define a subclass of δ-act, which we call 3-act. In other words, a three-way operator implicitly defines a patch and delta functions. Although we formalize it, this fact has been known for quite some time. It is for instance exploited by RCS [Tichy, 1985] to provide selective merges via the `rcsmerge` command, equivalent (except in rare circumstances) to a `diff` followed by a `patch`.

5. A 3-act can be embedded in a merge monoid. Furthermore, if a three-way merge succeeds (the result is not a conflict), then it meets our operation-based definition of a conflict-free merge.

This result may seem surprising, since a three-way merge is presented by most authors as the prototypical state-based approach to merging, in (sharp) contrast to operation-based merging. The insight here is that many three-way merge algorithms can also be interpreted as a reconciling two sequences of operations which are automatically derived form the three states. Key to this connection is the notion of safe embedding. This connection extends the "cover" of our definition to an important class of merges typically described as state-based.

## 2 Main Body

### 2.1 Preliminaries

We recall some basic notions and facts from abstract algebra.

A monoid $(M, \cdot)$ is a set together with an associative binary operation that has an identity element.

Given a set $S$, the set of functions mapping $S$ to itself, $S^S := \{f : S \to S\}$ forms a monoid under function composition. These functions are called transformations or, from a category theoretic perspective, endomorphisms. Although the term (unary) operation is also justifiable from the perspective of universal algebra, we shall prefer the term transformation for the rest of this paper.

As matter of notation, we will write $(f\,;g)(x) := g(f(x))$ for function composition since it captures more naturally the historical order of a sequence of transformations.

Given a monoid $(M, \cdot)$, a submonoid is a subset $N \subseteq M$ that is closed under the monoid's operation: $\forall x, y \in N \Rightarrow x \cdot y \in N$.

For our purposes, it suffices to define a transformation monoid $(S, M)$ as a set $S$ together with a submonoid $M \subseteq S^S$, in other words, a subset of the transformations on $S$ that is closed under function composition. For a more general definition see Barr and Wells [1995, chap. 3.2].

### 2.2 The merge monoid

Let $\mathcal{Q}$ be a set of states, not necessarily finite. Let $\mathcal{Q}_\perp := \mathcal{Q} \cup \{\perp\}$ be the extension of $\mathcal{Q}$ with the "conflict" element $\perp \notin \mathcal{Q}$. Let $\mathcal{Q}_\perp^{\mathcal{Q}} := \{f : \mathcal{Q}_\perp \to \mathcal{Q}_\perp \mid f(\perp) = \perp\}$ be the set of all conflict-preserving transformations – endomorphisms on $\mathcal{Q}_\perp$ for which $\perp$ is a fixed point. The rationale for imposing this restriction is that we do not want "normal" transformations to be able to solve conflicts. Since $\mathcal{Q}_\perp^{\mathcal{Q}}$ is closed under function composition, it is a submonoid of the monoid of all transformations on $\mathcal{Q}_\perp$. Therefore, $(\mathcal{Q}_\perp, \mathcal{Q}_\perp^{\mathcal{Q}})$ meets the definition of a transformation monoid, and we shall call it *the full transformation monoid with conflicts*.

In applications we are typically interested only in certain classes of transformation, e.g. for text files we may be interested only in line insertions and deletions. Thus, it makes sense to consider a only the transformations from a submonoid $\mathcal{T}$ of $\mathcal{Q}_\perp^{\mathcal{Q}}$. Let $(\mathcal{Q}_\perp, \mathcal{T})$ be known as a *transformation monoid with conflicts* or, more succinctly, a *merge monoid*.

### 2.3 Conflict-free merges

Let $\alpha = (\alpha_m)$ and $\beta = (\beta_n)$ be strings (finite sequences) of functions from $\mathcal{T}$, formally elements of $\mathcal{T}^*$. A string of transformations is applied to an element $X \in \mathcal{Q}_\perp$ using the expected semantics of function composition: $\alpha(X) := (\alpha_1\,;\cdots;\alpha_m)(X)$. By convention the empty string $\varepsilon$ is applied as the identity function on $\mathcal{Q}_\perp^{\mathcal{Q}}$.

We can attempt a definition of a conflict-free merge in terms of commuting transformation sequences. Let $O, A$ and $B$ be three states from $\mathcal{Q}$, such that $A = \alpha(O)$ and $B = \beta(O)$. We can claim that $\{\alpha, \beta\}$ form a conflict-free merge at $O$ with the result $M$ iff $M = \alpha\beta(O) = \beta\alpha(O)$, or more explicitly

$$M = (\alpha_1\,;\cdots;\alpha_m\,;\beta_1\,;\cdots;\beta_n)(O) = (\beta_1\,;\cdots;\beta_n\,;\alpha_1\,;\cdots;\alpha_m)(O)$$

This tentative definition is unsatisfactory. The following example will illustrate its deficiency.

**Example 1.** Let $U = \{a, b, \ldots\}$ be a universe of "atoms", and let the states be subsets of $U$, i.e. $\mathcal{Q} = \wp(U)$. We define a family[8] of transformations, inspired from the behavior of the `patch` program, that successfully insert and respectively remove an atom $a$ from a state $S \in \mathcal{Q}$ iff $a$ is and respectively is not included in $S$, producing a conflict otherwise:

$$\mathfrak{i}_a(S) := \begin{cases} S \cup \{a\} & \text{if } a \notin S \\ \perp & \text{if } a \in S \end{cases} \qquad \mathfrak{d}_a(S) := \begin{cases} S \setminus \{a\} & \text{if } a \in S \\ \perp & \text{if } a \notin S \end{cases}$$

The above two transformations also need to map $\perp$ to $\perp$ in order to be elements of $\mathcal{Q}_\perp^{\mathcal{Q}}$. Unless otherwise specified, assume this mapping to happen in all examples.

[6] It should not be construed that these are sufficient for a *useful* three-way merge algorithm. They are only required properties.

[7] Used in RCS [Tichy, 1985], CVS [Berliner, 1990] and many other revision control systems.

[8] A family of this kind can be defined as the set of all partial applications of a two-argument function.

Observe that if $a \notin S$, then $(\mathfrak{i}_a ; \mathfrak{d}_a)(S) = S$. Now consider merging the sequences of operations $\alpha = (\mathfrak{i}_a ; \mathfrak{d}_a)$ and $\beta = \mathfrak{i}_a$ starting from the initial state $O = \varnothing$. We obtain:

$$\alpha\beta(O) = (\mathfrak{i}_a ; \mathfrak{d}_a ; \mathfrak{i}_a)(\varnothing) = \{a\} \neq \beta\alpha(O) = (\mathfrak{i}_a ; \mathfrak{i}_a ; \mathfrak{d}_a)(\varnothing) = \bot$$

It is unsatisfactory that we have to declare a conflict when the sequence $\mathfrak{i}_a ; \mathfrak{d}_a$ is equivalent to the identity function when applied to $O$. One can interpret the above result as a paradox[9] in the following way: more information about the operations performed (or equivalently about intermediate states) leads to more conflicts, not fewer as we would hope.

In order to avoid this paradox, we attempt to give a definition that does not use the exact sequences $\alpha$ and $\beta$, but rather any other pair of transformations $\dot{\alpha}$ and $\dot{\beta}$ from $\mathcal{T}$ which have the same observable effect[10] on $O$, more precisely $\dot{\alpha}(O) = \alpha(O)$ and $\dot{\beta}(O) = \beta(O)$, but unlike the original sequences have the advantage that they commute at $O$ producing the result of the merge $M = \dot{\alpha}\dot{\beta}(O) = \dot{\beta}\dot{\alpha}(O)$. To continue our example, we can choose $\dot{\alpha} = \varepsilon$ and $\dot{\beta} = \mathfrak{i}_a$ and get result we'd expect $M = \mathfrak{i}_a(O)$.

Unfortunately, by fixing only two points for $\dot{\alpha}$ and $\dot{\beta}$, we've relaxed the definition too much, as we shall argue in the following example:

**Example 2.** Let $\mathcal{T} = \mathcal{Q}_\bot^{\mathcal{Q}}$ and we want to merge $\alpha = \mathfrak{i}_a$ with $\beta = \mathfrak{i}_b$ starting from $O = \varnothing$. We can choose the following two transformations as "solution":

$$\varphi(S) = \begin{cases} \{a\} & \text{if } S = \varnothing \\ \varnothing & \text{otherwise} \end{cases} \qquad \psi(S) = \begin{cases} \{b\} & \text{if } S = \varnothing \\ \varnothing & \text{otherwise} \end{cases}$$

Trivially $\alpha(O) = \varphi(O) = \{a\}$ and $\beta(O) = \psi(O) = \{b\}$. Since $\psi(\varphi(O)) = \varphi(\psi(O)) = \varnothing$, according to our tentative definition, we'd have to accept that $\varnothing$ is a valid result for the merge. Clearly, we'd be hard pressed to sell this to a discerning user.

Intuitively, we need to coerce the $\dot{\alpha}$ and $\dot{\beta}$ to have the same effect as $\alpha$ and $\beta$ not only at $O$ but also for any argument that is "$O$-like". With this additional requirement, $\varphi$ would not be able to "impersonate" $\alpha$ at $\{b\}$ in the above example.

We now formalize "$O$-like" in a way that allows the useful choices for $\dot{\alpha}$ and $\dot{\beta}$ that we made in example 1, but disallowing the "solution" for example 2.

**Definition 1** (safe embedding)**.** Let $D \subseteq \mathcal{Q}_\bot$ be a subset of the domain of transformations. A transformation $f$ is safely embedded in another transformation $g$ over $D$, and we write $f \sqsubseteq_D g$, iff

$$\forall x \in D, f(x) \neq \bot \;\Rightarrow\; g(x) = f(x)$$

Informally, $g$ safely embeds $f$ over $D$ if $g$ is a "rewriting" of $f$ that "breaks less often" over $D$. If $g$ safely embeds $f$ on the entire $\mathcal{Q}$, we can unambiguously write $f \sqsubseteq g$.

[9] The (general) paradox of operation-based merging.

[10] Depending on how loose of an analogy you wish to admit, you can see these functions as approximations, or even "weak bisimulations".

We have already seen both examples and counterexamples of safe embeddings. In example 1, $(\mathfrak{i}_a ; \mathfrak{d}_a) \sqsubseteq \varepsilon$. In example 2 however, $\mathfrak{i}_a(\{b\}) = \{a, b\} \neq \varphi(\{b\}) = \varnothing$, thus $\mathfrak{i}_a \not\sqsubseteq \varphi$.

Recall that given a function $f : X \to Y$, the preimage of a subset of the codomain $B \subseteq Y$, written[11] $f^{-1}[B]$ is the subset of the domain $\{x \in X \mid f(x) \in B\}$. Also recall that the equaliser of two functions $f$ and $g$ both defined on the same domain $X$ and codomain $Y$ is the subset of the domain $\mathrm{Eq}(f, g) := \{x \in X \mid f(x) = g(x)\}$. Using these notions, we can give the following trivial characterization of safe embedding:

**Proposition 1.** For all transformations $\alpha, \beta$:

$$\alpha \sqsubseteq_D \beta \iff \alpha^{-1}[\mathcal{Q}] \cap D \subseteq \mathrm{Eq}(\alpha, \beta)$$
$$\alpha \sqsubseteq \beta \iff \alpha \sqsubseteq_{\alpha^{-1}[\mathcal{Q}]} \beta \iff \alpha^{-1}[\mathcal{Q}] \subseteq \mathrm{Eq}(\alpha, \beta)$$
$$\alpha \sqsubseteq_D \beta \text{ and } \beta \sqsubseteq_D \alpha \iff D \subseteq \mathrm{Eq}(\alpha, \beta)$$

Now observe the following: we are only interested in merging states from $\mathcal{Q}$, in other words elements of $\mathcal{Q}_\bot$ other than $\bot$. If $\alpha(O) = A$ and $\beta(O) = B$, then if we impose the condition that $A, B \in \mathcal{Q}$, we obtain a restriction on what "$O$-like" states can be, which can be expressed in terms of preimages as $O \in \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}]$.

**Definition 2** (conflict-free merge)**.** Let $\alpha, \beta \in \mathcal{T}$. A conflict-free merge of $\alpha$ and $\beta$ is a pair of transformations $\dot{\alpha}, \dot{\beta} \in \mathcal{T}$ with the following properties:

i) $\alpha \sqsubseteq \dot{\alpha}$ and $\beta \sqsubseteq \dot{\beta}$

ii) $\forall X \in \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}],\ \dot{\beta}(\dot{\alpha}(X)) = \dot{\alpha}(\dot{\beta}(X)) \neq \bot$

The first requirement in the above definition cannot be relaxed "even one bit" without admitting solutions such as the one from example 2. To illustrate this, assume that we'd replace (i) with $\alpha \sqsubseteq_C \dot{\alpha}$ and $\beta \sqsubseteq_C \dot{\beta}$, where $C = \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}]$. This may seem innocuous enough because $C$ is exactly the set over which $X$ varies in the second requirement.

explain me better

**Example 3.** Let $\mathcal{T} = \mathcal{Q}_\bot^{\mathcal{Q}}$ and we want to merge $\alpha = \mathfrak{i}_a$ with $\beta = \mathfrak{i}_b$ starting from $O = \varnothing$ (same as example 2). The proposed solution is:

$$\varphi(S) = \begin{cases} S \cup \{a\} & \text{if } \{a, b\} \notin S \\ \varnothing & \text{otherwise} \end{cases}$$
$$\psi(S) = \begin{cases} S \cup \{b\} & \text{if } \{a, b\} \notin S \\ \varnothing & \text{otherwise} \end{cases}$$

Observe that $C = \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}] = \{S \mid \{a, b\} \notin S\}$ and $\alpha \sqsubseteq_C \varphi$ and also $\beta \sqsubseteq_C \psi$. But $\psi(\varphi(\varnothing)) = \psi(\{a\}) = \varnothing$ and $\varphi(\psi(\varnothing)) = \varphi(\{b\}) = \varnothing$, which is again difficult to accept as a merge of two insertions. On the other hand $\alpha \not\sqsubseteq \varphi$ (and $\beta \not\sqsubseteq \psi$), so $\varphi$ and $\psi$ do not constitute a merge according to definition 2.

[11] We use the less often encountered notation with square brackets because most examples involve sets as function arguments.

**Open Question 1.** What happens if we replace (ii) in definition 2 with a "point-based" property: "Given $\alpha, \beta$ and $O \in \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}]$ a conflict-free merge is $\dot{\alpha}$ and $\dot{\beta}$ s.t. (i) holds and $\dot{\beta}(\dot{\alpha}(O)) = \dot{\alpha}(\dot{\beta}(O)) \neq \bot$". Is this equivalent to definition 2 or not?

### 2.4 Solving false conflicts

Relying just on conflict-free merges is not entirely satisfactory. For instance, the merge of $\alpha = \mathfrak{i}_a$ with $\beta = \mathfrak{i}_a$ does not admit a conflict-free solution because we cannot conceivably embed $\mathfrak{i}_a$ in anything but itself, and $(\mathfrak{i}_a ; \mathfrak{i}_a)(S) = \bot$ for any $S$.

The solution may seem blatantly obvious in this case: keep just one $\mathfrak{i}_a$. Unfortunately, we cannot easily generalize this approach. Consider the following counterexample where $\mathcal{Q} = \mathbb{N}$ and $\mathrm{succ}(n) := n + 1$. If $\alpha = \beta = \mathrm{succ}$, then there is a conflict-free merge, namely $(\mathrm{succ}; \mathrm{succ})$. Eliminating the duplicate succ here does not seem justifiable. We note here that using Lippe and Oosterom's weave concept, the merge would be just succ.

fix spacing around ⇊

Let $\Yup$ be the transformation that always results in $\bot$, i.e. $\forall X \in \mathcal{Q}, \Yup(X) = \bot$. Note that $\Yup$ is an absorbing element[12], meaning that for any transformation $\alpha$, $(\alpha; \Yup) = (\Yup ; \alpha) = \Yup$.

**Definition 3** (zeropotent[13] transformation)**.** A transformation $\alpha$ is called zeropotent iff $(\alpha ; \alpha) = \Yup$.

As examples, both $\mathfrak{i}_a$ and $\mathfrak{d}_a$ are zeropotent, while succ is not zeropotent.

It may seem that if a transformation is zeropotent we may always eliminate duplicates. This is not exactly true. Even if $\alpha$ is zeropotent, nothing can be said in general about the sequence $(\alpha;\beta;\alpha)$. For instance if $\alpha = \mathfrak{i}_a$ and $\beta = \mathfrak{d}_a$ then $(\mathfrak{i}_a;\mathfrak{d}_a;\mathfrak{i}_a)(\varnothing) = \{a\} \neq \bot$.

Simply discarding duplicate zeropotent operations would result in a surprising change of result for the merge in example 1, where $\alpha = (\mathfrak{i}_a;\mathfrak{d}_a)$ and $\beta = \mathfrak{i}_a$. If it were permissible to discard a duplicate $\mathfrak{i}_a$, the merge result could be just $(\mathfrak{i}_a ; \mathfrak{d}_a)(\varnothing) = \varnothing$.

this needs better explanations

We argue that merges such as the above are invalid, because they lead to the following paradox: a transformation such as $\mathfrak{d}_a$ from $\beta$ can "cancel out" transformations that do not necessarily precede it, i.e. $\mathfrak{i}_a$ from $\beta$. In the typical scenario of lost updates, the *last* update wins (and the other are lost). In this incorrect merging example however, an update that may not even have happened, namely $\mathfrak{i}_a$ from $\beta$, is guaranteed to be lost upon the merge. That's why we call this scenario a lost *future* update.

cite it somehow

Unfortunately, some theoretical papers and practical systems define merges in a way that leads to the paradox of lost future updates. Lippe and van Oosterom [1992] perform merges using the weave concept which, by performing unification first, results in the incorrect $\varnothing$ solution for the merge in example 1. We have verified that the revision control system darcs also suffers from this problem. Example 1 can be easily translated into a text merging problem by mapping each atom to a text line.

We can easily generalize the notion of zeropotency to a pair of transformations in the following way:

**Definition 4** (mutually zeropotent transformations)**.** Two transformations $\alpha$ and $\beta$ are called mutually zeropotent iff $(\alpha ; \beta) = (\beta ; \alpha) = \Yup$.

Perhaps counterintuitively, one can find a pair of transformations that are mutually zeropotent, but still can be merged conflict-free (using safe embeddings of course): $\alpha = (\mathfrak{i}_a;\mathfrak{d}_a;\mathfrak{i}_b)$ and $\beta = (\mathfrak{i}_b ; \mathfrak{d}_b ; \mathfrak{i}_a)$.

An immediate consequence of the above observation is that we cannot hope to give a rule for when it is "safe" to eliminate duplicate transformations solely based on the zeropotency relation between the initial sequences of transformations we want to merge. Let us further illustrate the difficulty of the matter with the following:

**Example 4.** Let $\alpha = (\mathfrak{i}_a ; \mathfrak{i}_b ; \mathfrak{d}_a ; \mathfrak{i}_a)$ and $\beta = (\mathfrak{i}_b ; \mathfrak{i}_a)$. Clearly, no conflict-free merge exists due to $\mathfrak{i}_b$. What is less clear is how to give definition of false conflicts that would allow a duplicate $\mathfrak{i}_b$ to be removed, while disallowing any $\mathfrak{i}_a$ from being removed.

**Definition 5** (merge with safe prefix false-conflict elmination)**.** Let $\alpha$, $\beta$ be transformations we want to merge. A merge with safe prefix false-conflict elimination is a triplet of transformations $\dot{\alpha}$, $\dot{\beta}$ and $\dot{\gamma}$ with the following properties:

i) $\alpha \sqsubseteq (\dot{\gamma} ; \dot{\alpha})$ and $\beta \sqsubseteq (\dot{\gamma} ; \dot{\beta})$

ii) $\forall X \in \alpha^{-1}[\mathcal{Q}] \cap \beta^{-1}[\mathcal{Q}]$, $(\dot{\gamma};\dot{\alpha};\dot{\beta})(X) = (\dot{\gamma};\dot{\beta};\dot{\alpha})(X) \neq \bot$

iii) $\forall \dot{\gamma}_1, \dot{\gamma}_2 \in \mathcal{T}$, $(\dot{\gamma}_1 ; \dot{\gamma}_2) = \dot{\gamma} \Rightarrow (\dot{\gamma}_2 ; \dot{\alpha})$ and $(\dot{\gamma}_2 ; \dot{\beta})$ do not admit a conflict free merge.

need a decent phrase here

The purpose of (iii) in the above definition is to prevent "pulling the zipper too much". In example 4 for instance, it allows us to reject the following solution $\dot{\gamma} = (\mathfrak{i}_a;\mathfrak{i}_b)$, $\dot{\alpha} = (\mathfrak{d}_a;\mathfrak{i}_a)$ and $\dot{\beta} = \varepsilon$, because we can choose $\dot{\gamma}_1 = \mathfrak{i}_b$ and $\dot{\gamma}_2 = \mathfrak{i}_a$ with the result that $(\dot{\gamma}_2 ; \dot{\alpha}) = (\mathfrak{i}_a ; \mathfrak{d}_a ; \mathfrak{i}_a)$ and $(\dot{\gamma}_2 ; \dot{\beta}) = \mathfrak{i}_a$ admit a conflict free merge.

**Open Question 2.** One could give a definition where the duplicated $\dot{\gamma}$ lies "in the middle", meaning that $\alpha \sqsubseteq (\dot{\alpha}_1 ; \dot{\gamma} ; \dot{\alpha}_2)$ and similar for $\beta$. Can this be reduced to definition 5 or not?

### 2.5 Dependencies

**Definition 6.** We say that $\beta$ depends on $\alpha$ over $D \subseteq \mathcal{Q}$ and write $\alpha \prec_D \beta$ iff the following three statements are all true:

i) $\forall x \in D,\ \alpha(x) \neq \bot \Rightarrow \beta(\alpha(x)) \neq \bot$

ii) $\forall x \in D,\ \beta(x) = \bot \Rightarrow \alpha(x) \neq \bot$

iii) $\exists x \in D,\ \beta(x) = \bot$

The intuition is that $\alpha$ makes $\beta$ "not break" over $D$ if it is applied before it. The condition (iii) is necessary to prevent (ii) from holding vacuously. Without (iii) the definition would imply that the identity transformation $\varepsilon$ depends on any other

[12] Some authors use the term zero element.

[13] For the algebraically interested reader, we note that if all transformations from $\mathcal{T}$ are zeropotent, then function composition itself becomes a zeropotent operation in the sense used in abstract algebra, more precisely: $xxy = yyx = xx$.

transformation $\alpha$ because $\alpha(x) \neq \bot$ implies that $\varepsilon(\alpha(x)) = \alpha(x) \neq \bot$, which satisfies (i), and since $\varepsilon(x) \neq \bot$ for any $D \subseteq \mathcal{Q}$ and any $x \in D$, (ii) holds vacuously. In intuitive terms, there must be some "danger" that $\beta$ is exposed to before $\alpha$ can "claim to make it safe".

Henceforth we will make the notation $\alpha \rightsquigarrow_D \beta$ iff $\forall x \in D,\ \alpha(\beta(x)) = \beta(\alpha(x))$.

**Proposition 2.** $\alpha \rightsquigarrow_D \beta$ and $\alpha \prec_D \beta$ are mutually exclusive.

*Proof.* Assume they are both true at the same time. Then by definition 6(iii) we can choose $z \in D$ s.t. $\beta(z) = \bot$ and by (ii) $\alpha(z) \neq \bot$, which by (i) entails that $\beta(\alpha(z)) \neq \bot$.

On the other hand, $\alpha \rightsquigarrow_D \beta$ implies that $\beta(\alpha(z)) = \alpha(\beta(z)) = \alpha(\bot) = \bot$. Contradiction! □

**Corollary 1.** (dependencies preclude commutation and vice versa)

$$\alpha \prec_D \beta \text{ or } \beta \prec_D \alpha \Rightarrow \alpha \not\rightsquigarrow_D \beta$$
$$\alpha \rightsquigarrow_D \beta \Rightarrow \alpha \not\prec_D \beta \text{ and } \beta \not\prec_D \alpha$$

## 3 Delta-acts

Let $\mathcal{Q}$ and $\mathcal{D}$ be a sets. The elements of $\mathcal{Q}$ are interpreted as states and the elements of $\mathcal{D}$ as deltas between states.

**Definition 7.** (δ-act) A δ-act is a quadruple $(\mathcal{Q}, \mathcal{D}, \cdot, +)$ where $\cdot\colon \mathcal{Q} \times \mathcal{Q} \to \mathcal{D}$ is the delta creation function (think `diff`), and $+\colon \mathcal{Q} \times \mathcal{D} \to \mathcal{Q} \cup \{\bot\}$ is the delta application function (think `patch`), subject to the following law: $\forall A, B \in \mathcal{Q},\ A + AB = B$.

The only relationship imposed on $\cdot$ and $+$ is that the delta from $A$ to $B$, when applied to $A$ must yield exactly $B$. In general we can impose no other requirement. In particular we cannot say anything about "how" a delta gets applied to a state other than the one it was based on ($A$). Note that delta application may "fail", i.e. produce $\bot$.

**Definition 8.** (three-way merge operator) A three way merge operator is a ternary operator with signature $: \mathcal{Q} \times \mathcal{Q} \times \mathcal{Q} \to \mathcal{Q} \cup \{\bot\}$, written in infix notation as $A \llcorner O \lrcorner B$, where $O$ is the "reference" state, while $A$ and $B$ are the states "to be merged", subject to the following two laws:

| | |
|---|---|
| $A \llcorner A \lrcorner B = B$ | identity law |
| $A \llcorner O \lrcorner B = B \llcorner O \lrcorner A$ | tip-commutativity |

We will show how a three-way merge operator implicitly defines a δ-act. First, the set of deltas $\mathcal{D}$ is the set of transformations from $\mathcal{Q}$ to $\mathcal{Q}_\bot$. The delta creation function maps any pair of states to a function. Naturally, this is defined by partial application of the three way merge operator: $AB \mapsto (X \mapsto X \llcorner A \lrcorner B)$. The patch function is simply function application. We still need to verify the law of a δ-act. It holds due to the identity law of the three way merge operator:

$$A + AB = (X \mapsto X \llcorner A \lrcorner B)(A) = A \llcorner A \lrcorner B = B$$

Thus, we have shown that three-way merge operator defines a subclass of δ-act, which we call 3-act.

Although a 3-act is not exactly a merge monoid because the domain of the delta functions is just $\mathcal{Q}$, it can be trivially embedded in a merge monoid by adding a mapping from $\bot$ to $\bot$ for all delta functions.[14]

The question is now whether a three-way merge has any relationship to our definition of operation-based mering. Assume a three-way merge of $A \llcorner O \lrcorner B = M$ produces a valid state, i.e. $M \neq \bot$. The three-way merge also defines two transformations $\alpha(X) = X \llcorner O \lrcorner B$ and $\beta(X) = X \llcorner O \lrcorner A$. Observe that:

$$\alpha(\beta(O)) = (A \llcorner O \lrcorner O) \llcorner O \lrcorner B = (O \llcorner O \lrcorner A) \llcorner O \lrcorner B = A \llcorner O \lrcorner B$$
$$\beta(\alpha(O)) = (O \llcorner O \lrcorner B) \llcorner O \lrcorner A = B \llcorner O \lrcorner A = A \llcorner O \lrcorner B$$

Thus, the two transformations commute, which means they satisfy definition 2 (conflict-free merge). In summary, we have shown that a conflict-free three-way merge satisfies our definition of an operation-based conflict-free merge. The "trick" here is that the two operations that are merged are implicitly defined.

[14] Alternatively, we could have defined the three way merge operator on $\mathcal{Q}_\bot$, and added an additional law that made the result $\bot$ if any of the three arguments are $\bot$. This construction seems less natural.

## References


Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, Englewood Cliffs, 1995. ISBN 0133238091.

Brian Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association. URL citeseer.ist.psu.edu/berliner90cvs.html.

Anne-Marie Kermarrec, Antony Rowstron, Marc Shapiro, and Peter Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC '01: Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*, pages 210–218, New York, NY, USA, 2001. ACM. ISBN 1-58113-383-9. doi: http://doi.acm.org/10.1145/383962.384020.

Sanjeev Khanna, Keshav Kunal, and Benjamin C. Pierce. A formal investigation of Diff3. In Arvind and Prasad, editors, *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, December 2007.

Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/359545.359563.

Ernst Lippe and Norbert van Oosterom. Operation-based merging. In *SDE 5: Proceedings of the fifth ACM SIGSOFT symposium on Software development environments*, pages 78–87, New York, NY, USA, 1992. ACM. ISBN 0-89791-554-2. doi: http://doi.acm.org/10.1145/142868.143753.

Norman Ramsey and Előd Csirmaz. An algebraic approach to file synchronization. *SIGSOFT Softw. Eng. Notes*, 26(5):175–185, 2001. ISSN 0163-5948. doi: http://doi.acm.org/10.1145/503271.503233.

Jonathan D. H. Smith. *Mal'cev Varieties*, volume 554 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin/New York, 1976. ISBN 9783540079996. doi: 10.1007/BFb0095447.

Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1):63–108, 1998. ISSN 1073-0516. doi: http://doi.acm.org/10.1145/274444.274447.

D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*, pages 172–182, New York, NY, USA, 1995. ACM. ISBN 0-89791-715-4. doi: http://doi.acm.org/10.1145/224056.224070.

Walter F. Tichy. RCS — a system for version control. *Software — Practice and Experience*, 15(7):637–654, 1985. doi: http://dx.doi.org/10.1002/spe.4380150703. URL citeseer.ist.psu.edu/tichy85rcs.html.