Algebra of Data Reconciliation

Elod P. Csirmaz* and Laszlo Csirmaz**

Abstract—**With distributed computing and mobile applications becoming ever more prevalent, synchronizing diverging replicas of the same data is a common problem. Reconciliation – bringing two replicas of the same data structure as close as possible without overriding local changes – is investigated in an algebraic model. Our approach is to consider two sequences of simple commands that describe the changes in the replicas compared to the original structure, and then determine the maximal subsequences of each that can be propagated to the other. The proposed command set is shown to be functionally complete, and an update detection algorithm is presented which produces a command sequence transforming the original data structure into the replica while traversing both simultaneously. Syntactical characterization is provided in terms of a rewriting system for semantically equivalent command sequences. Algebraic properties of sequence pairs that are applicable to the same data structure are investigated. Based on these results the reconciliation problem is shown to have a unique maximal solution. In addition, syntactical properties of the maximal solution allows for an efficient algorithm that produces it.**

Keywords—**file synchronization, algebraic model, confluence, rewriting system.**

MSC classes—**08A02, 08A70, 68M07, 68P05.**

ACM classes—**D.4.3, E.5, F.2.2, G.2.**

I. Introduction

Synchronization of diverging copies of some data stored on several independent devices is a mechanism we nowadays take for granted. Examples are accessing and editing calendar events, documents, spreadsheets, or distributed internet and web services hosted in the cloud. While there are numerous commercially available synchronization tools for, for example, the case of filesystems [6], [7], [8], [11], [12], there have been little progress on their theoretical foundation. The main aim of this paper is to provide such a mathematical framework and investigate its properties. Our synchronization paradigm follows the one described in [2] and depicted on Figure 1. Two identical replicas of the original data structure Φ are updated independently yielding the diverged copies Φ_1 and Φ_2. The *update detector* compares the original (Φ) and current (Φ_1 or Φ_2) state, and extracts an update information describing the differences between the original and the replica. The *reconciler* uses this information to propagate as many of the changes to the other replica as possible without destroying local changes. The remaining updates are *conflicts* which are handled by a separate conflict resolver typically requiring human intervention. The final conflict resolution is outside of the scope of this paper, as it should handle contradictory updates to the replicas that cannot be resolved automatically based on the information available.

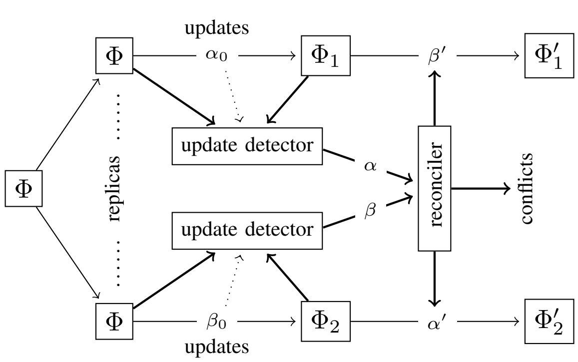

Fig. 1. The synchronization process

In terms of the usual classification of synchronizers (see, e.g., [8], [10], [11]), our approach is an operation-based one as changes in the data structures are modeled as the effects of specific command sequences. The task of the update detector is to produce such (normalized) command sequences, and that of the reconciliation algorithm is to identify the commands which can be propagated to the other replica. The central problem during both update detection and reconciliation is the ordering (scheduling) of these commands. If update detection is based on comparing the original state of the system to its new state, then one can easily collect a set of commands that can create the differences, but they must be ordered (if at all possible) in a way that the sequence can be applied to the initial system without causing an error. Similarly, during reconciliation, typically many maximal propagable command sequences exist. For more information, we refer to the excellent survey of the so-called optimistic replication algorithms [10], and, as working examples, to IceCube, where multiple orders are tested to find an acceptable one [6], [7], or Bayou, where reconciling updates happens by redoing them in a globally determined order [12].

This paper continues and extends the investigation started in [5]. An important contribution of [5] is the abstract notion of hierarchical data structures and data manipulating commands which, on the one hand, faithfully represents real data structures and typical commands on them, and, on the other, has a rich and intriguing algebraic structure. This structure is investigated in a more general setting here, while additional structural results and characterizations are provided. Based on these theorems it is proved that even in the extended setting the reconciliation problem has a unique maximal solution in a very strong sense, and that the solution can be found using a simple and efficient algorithm.

While the exposition uses the specific terminology of filesystems and filesystem commands, the results can form the basis of treating the synchronization of other structured data stores in an algebraic fashion, with similarly strong theoretical

*e-mail: elod@epcsirmaz.com
**e-mail: csirmaz@renyi.hu Rényi Insititute, Budapest, and UTIA, Prague.

13 Oct 2021 01:56:14 PDT
211013-Csirmaz Version 1 - Submitted to Stud. Sci. Math. Hung.

foundations.

The paper is organized as follows. Hierarchical data structures and commands on which the algebraic setting is based are defined in Section II as *filesystems* and *filesystem commands*. Typical "real" filesystems and filesystem commands can be easily modeled using these abstract notions. The suggested command set is complete in the sense that any filesystem can be transferred into any other one by an appropriately chosen command sequence. Command pairs are considered in Section III where a set of (syntactical) rewriting rules are defined. These rules define a syntactical consequence relation among command sequences (one sequence can be derived from the other one using the rewriting rules), which is shown to coincide with the semantic consequence relation for an important subset of command sequences. This subset, called *simple sequences*, is introduced in Section IV where additional properties are stated and proved. The update detection algorithm is detailed in Section V. In reference to rewriting systems, sequence pairs applicable to the same filesystem are called *refluent*. Understanding their structure requires significant effort. Section VI proves several partial results which are used in Section VII where the uniqueness of the maximal reconciliation is proved together with the correctness of the algorithm identifying it. Finally Section VIII summarizes the results and lists some open problems.

II. Definitions

Informally, a *filesystem* is modeled as a function which populates some fixed virtual namespace $\mathbb{N}$ with values from a set $\mathbb{V}$. Elements of this virtual namespace $\mathbb{N}$ are *nodes*, which are arranged into a tree-like structure reflecting the hierarchical structure of the namespace. A filesystem assigns values to each virtual node such that along each path starting from a root there are finitely many directories, followed by an optional file value, followed by empty values. This virtual namespace can actually reflect the (fully qualified) names of all imaginable files and directories. In this case, renaming a file (or a directory) means that the contents of the file (or the whole directory subtree) is moved from one location to another.

A. Node structure

Formally, we fix an arbitrary and possibly infinite *node structure* $\mathbb{N}$ endowed with the partial function $\uparrow : \mathbb{N} \to \mathbb{N}$ which returns the parent of every non-root node (it is not defined on roots of which there might be several). This function must induce a tree-like structure, which means that there must be no loops or infinite forward chains. We say that n *is above* m and write $n \prec m$ if $n = \uparrow^i(m)$ for some $i \ge 1$. As usual, $n \preceq m$ means $n \prec m$ or $n = m$, which is a partial order on $\mathbb{N}$. Minimal elements of this partial order are the roots of $\mathbb{N}$. The nodes n and m are *comparable* if either $n \preceq m$ or $m \preceq m$, and they are *uncomparable* or *independent*, written as $n \,\|\, m$, otherwise.

B. Filesystem values, filesystems

A filesystem populates the nodes with values from a set $\mathbb{V}$ of possible filesystem values. $\mathbb{V}$ is partitioned into directory, file, and empty values as $\mathbb{V} = \mathbb{D} \cup \mathbb{F} \cup \mathbb{O}$. For a value $v \in \mathbb{V}$ its *type*, denoted by $\mathtt{tp}(v)$, is the partition it belongs to, thus it is one of $\mathbb{D}$, or $\mathbb{F}$ or $\mathbb{O}$. A *filesystem* is a function $\Phi : \mathbb{N} \to \mathbb{V}$ which has the *tree property*: along any branch starting from a root there are zero or more directory values, then zero or one file value which is followed by empty values only.[1] We assume that neither $\mathbb{D}$ nor $\mathbb{O}$ is empty, and $\mathbb{F}$ has at least two elements. The collection of all filesystems is denoted by $\mathcal{X}$.

C. Filesystem commands

The set of available filesystem commands will be denoted by Ω, and the application of a command $\sigma \in \Omega$ to the filesystem $\Phi \in \mathcal{X}$ will be written as the left action $\sigma\Phi$. If σ is not applicable to Φ then we say that σ *breaks* Φ, which is denoted by $\sigma\Phi = \bot$. Thus filesystem commands are modeled as functions mapping $\mathcal{X}$ into $\mathcal{X} \cup \{\bot\}$, where $\bot$ indicates failure.

The actual command set is specified so that, on one hand, it reflects the usual filesystem commands, and, on the other, it is more symmetric and more uniform. It is well-known that the actual choice of the command set has profound impact on whether reconciliation is possible or not, see [4], [9], and one of the main contributions of [5] is the systematic symmetrization of the traditional filesystem command set. Accordingly, commands in Ω are represented by triplets specifying

- a node $n \in \mathbb{N}$ on which the command acts,
- a precondition which specifies the type of the value the filesystem must have at node n before executing the command, and
- a replacement value to be stored at n.

Such a command is applicable if the precondition holds, and preforming the replacement does not destroy the tree property of the filesystem.

Formally, the commands in Ω are the triplets $\sigma = \langle n, t, x \rangle$ where $n \in \mathbb{N}$ is a node, $t \in \{\mathbb{D}, \mathbb{F}, \mathbb{O}\}$ specifies the precondition by requesting the current filesystem value at n to have type t, and $x \in \mathbb{V}$ is the replacement value. The *input* and *output type* of the command $\sigma = \langle n, t, x \rangle$ is t and $\mathtt{tp}(x)$, respectively. The effect of the command $\langle n, t, x \rangle$ on the filesystem $\Phi \in \mathcal{X}$ is defined as

$$\langle n, t, x \rangle \Phi = \begin{cases} \Phi_{[n \to x]} & \text{if } \mathtt{tp}(\Phi(n)) = t \text{ (precondition)} \\ & \quad \text{and } \Phi_{[n \to x]} \in \mathcal{X} \text{ (tree property);} \\ \bot & \text{otherwise,} \end{cases}$$

where the operator $\Phi_{[n \to x]}$ changes the value of the function Φ only at n to x as

$$\Phi_{[n \to x]}(m) = \begin{cases} x & \text{if } m = n, \\ \Phi(m) & \text{otherwise.} \end{cases}$$

[1] This definition allows filesystems with infinitely many non-empty nodes, requiring only that every branch is eventually empty. The set $\mathbb{F}$ reflects all possible file contents together with additional metainformation. Also, the set $\mathbb{O}$ of "empty" values is not required to have a single element only.

If nodes encode the used namespace, then, for example, the typical filesystem command "*rmdir* n" corresponds to the abstract command $\langle n, \mathbb{D}, \mathsf{o}\rangle$ where $\mathsf{o} \in \mathbb{O}$ is some empty value; this command fails when either n is not a directory (input type mismatch), or the directory is not empty (after the replacement the tree property is violated). Similarly, editing a file at location n is captured by the command $\langle n, \mathbb{F}, \mathsf{f}\rangle$ where $\mathsf{f} \in \mathbb{F}$ represents the new file content.

Two additional filesystem commands, denoted by ϵ and λ, will be defined. In practice they do not naturally occur, and are not elements of Ω, but they are are useful when arguing about command sequences. The command ϵ breaks every filesystem while λ acts as identity: $\epsilon\Phi = \bot$ and $\lambda\Phi = \Phi$ for every $\Phi \in \mathcal{X}$. These commands have no nodes, or input or output types. Commands in Ω are denoted by σ, τ and ω.

D. Command categories

A command $\langle n, t, x\rangle$ matches the pattern $\langle n, \mathsf{T}, \mathsf{X}\rangle$ if $t \subseteq \mathsf{T}$ and $x \in \mathsf{X}$. Only data types $\mathbb{D}$, $\mathbb{F}$, $\mathbb{O}$, and their unions will be used in place of T and X with the union sign omitted. In a pattern the symbol $\cdot$ matches any value.

Depending on their input and output types commands can be partitioned into nine disjoint classes. *Structural commands* change the type of the stored data, while *transient commands* retain it. In other words, commands matching $\langle\cdot, \mathbb{O}, \mathbb{FD}\rangle$, $\langle\cdot, \mathbb{F}, \mathbb{OD}\rangle$, or $\langle\cdot, \mathbb{D}, \mathbb{OF}\rangle$ are structural commands, while those matching $\langle\cdot, \mathbb{F}, \mathbb{F}\rangle$, $\langle\cdot, \mathbb{O}, \mathbb{O}\rangle$, or $\langle\cdot, \mathbb{D}, \mathbb{D}\rangle$ are transient ones.

Structural commands are further split into *up* and *down* commands, where up commands "upgrade" the type from $\mathbb{O}$ to $\mathbb{F}$ to $\mathbb{D}$, while down commands "downgrade" the type of the stored value. That is, up commands are those matching $\langle\cdot, \mathbb{O}, \mathbb{FD}\rangle$ and $\langle\cdot, \mathbb{F}, \mathbb{D}\rangle$, while down commands are matching $\langle\cdot, \mathbb{D}, \mathbb{FO}\rangle$ and $\langle\cdot, \mathbb{F}, \mathbb{O}\rangle$. The type of the command $\sigma = \langle n, t, x\rangle$ is $\mathsf{tp}(\sigma) = \langle n, t, \mathsf{tp}(x)\rangle$, and then $\mathsf{tp}(\sigma) = \mathsf{tp}(\sigma')$ iff σ and σ' have the same node, same input type and same output type.

E. Command sequences

The free semigroup generated by Ω is Ω^*; this is the set of finite command sequences including the empty sequence λ. Elements of Ω^* act from left to right, that is, $(\alpha\sigma)\Phi = \sigma(\alpha\Phi)$ where $\sigma \in \Omega$ and $\alpha \in \Omega^*$. This definition is in full agreement with the definition of the special command λ. A command sequence α breaks a filesystem Φ if some initial segment of α breaks it. In other words, $(\sigma\alpha)\Phi = \bot$ if either $\alpha\Phi = \bot$ or $\sigma(\alpha\Phi) = \bot$. We use α, β and γ to denote command sequences.

We write $\alpha \sqsubseteq \beta$ to denote that β *semantically extends* α, that is, $\alpha\Phi = \beta\Phi$ for all filesystems Φ that α does not break. Similarly, $\alpha \equiv \beta$ denotes that α and β are *semantically equivalent*, meaning $\alpha\Phi = \beta\Phi$ for all $\Phi \in \mathcal{X}$. Clearly, $\alpha \equiv \beta$ if and only if both $\alpha \sqsubseteq \beta$ and $\beta \sqsubseteq \alpha$. As ϵ breaks every filesystem, $\alpha \not\equiv \epsilon$ means that α is defined on some filesystem. In this case we say that α is a *non-breaking sequence*.

For $\Delta \subseteq \Omega^*$ and $\alpha \in \Omega^*$, $\Delta \vDash \alpha$ denotes that for every filesystem Φ, if none of $\delta \in \Delta$ breaks Φ, then neither does α. This relation shares many properties of the "consequence" relation used in mathematical logic. As usual, $\Delta \vDash \Delta'$ means that $\Delta \vDash \delta$ for all $\delta \in \Delta'$, and we also write $\delta \vDash \alpha$ instead of $\{\delta\} \vDash \alpha$. The following claims are immediate from the definitions.

Proposition 1. (a) *If $\alpha \in \Delta$ then $\Delta \vDash \alpha$.* (b) *If $\Delta \vDash \alpha$ and $\Delta \subseteq \Delta'$, then $\Delta' \vDash \alpha$.* (c) *If $\Delta \vDash \Delta'$ and $\Delta' \vDash \alpha$ then $\Delta \vDash \alpha$.* (d) *$\Delta \vDash \alpha$ if and only if $\Delta' \vDash \alpha$ for some finite $\Delta' \subseteq \Delta$.* (e) *$\alpha\beta \vDash \alpha$.* (f) *If $\Delta \vDash \alpha$, then $\{\gamma\delta : \delta \in \Delta\} \vDash \gamma\alpha$.* □

Property (f) is an analog of the preconditioning property in logical systems. Observe that α is non-breaking iff $\alpha \not\vDash \epsilon$; and $\delta \sqsubseteq \alpha$ implies $\delta \vDash \alpha$ as the latter only requires that α is defined where δ is defined, while the former also requires that where they are both defined their effect is the same.

For two sequences $\{\alpha, \beta\} \not\vDash \epsilon$ iff there is a filesystem on which both α and β are defined. With an eye on rewriting systems [1], such a pair is called *refluent*.

F. Update detection and reconciliation

A command-based reconciliation system works with two command sequences α and β that have been applied to a single filesystem Φ yielding two different replicas Φ_1 and Φ_2 which we need to reconcile. While it is conceivable that the sequences are based on records of the executed filesystem operations, in several filesystem implementations no such records exist. In these cases the command sequences must be created by comparing Φ_1 (or Φ_2) to Φ. This process is called *update detection,* in which we also include transforming the resulting (or provided) command sequence into a canonical form required by reconciliation. Our first result is that the chosen set of filesystem commands has the required expressive power: one can always find a command sequence that transforms the original filesystem into the replica.

Theorem 2 (Informal, update detection)**.** (a) *Given arbitrary filesystems Φ_1 and Φ, there exists a canonical command sequence α such that $\Phi_1 = \alpha\Phi$. The commands in α can be found by traversing Φ and Φ_1 while searching for different node content; the order of the commands can be found in quadratic time in the length of α.*
(b) *Given any command sequence that transforms Φ to Φ_1, the corresponding canonical sequence can be created from it in quadratic time.* □

Reconciliation is the process of merging the diverging replicas as much as possible, and mark cases where it is not possible to do so without further—usually human—input as *conflicts.* In our approach it means applying as many updates (commands) as possible that have been applied to one replica to the other without breaking the filesystem or overriding local changes. The remaining commands are marked as conflicting updates. Resolving these conflicts, as they require knowledge and input not available in the filesystems, is outside the scope of the reconciliation algorithm. More formally, β' is a *reconciler for α over β* if

- β' consists of commands from β;
- for any Φ, β' is applicable to $\alpha\Phi$ whenever both $\alpha\Phi$ and $\beta\Phi$ are defined;

- no command in β' overrides the effect of any command from α.

The main result of this paper is that in this algebraic framework the reconciliation problem has a unique maximal solution in a very strong sense.

Theorem 3 (Informal, reconciliation)**.** *Suppose α and β are canonical sequences, and there is at least one filesystem neither of them breaks. Then there is a maximal reconciler β' for α over β which can be created from α and β in quadratic time. Moreover, β' is optimal in a very strong sense: for any sequence β'' consisting of commands of β, if β'' contains a command not in β', then either β'' overrides a change made by α, or $\alpha\beta''$ breaks every filesystem.* □

Theorem 2 is proved in Section V as Theorems 19 and 20, while Theorem 3 follows from Theorem 28 in Section VII and the discussion following its proof.

III. Rules on command pairs

Proposition 4 (Command pairs 1)**.** *Suppose $\sigma, \tau \in \Omega$ are on the same node n. Then exactly one of the following possibilities hold:*
(a) *$\sigma\tau \sqsubseteq \omega$ for some $\omega \in \Omega$ also on node n,*
(b) *$\sigma\tau \equiv \epsilon$.*

Proof. Let the two commands be $\sigma = \langle n, t, x\rangle$ and $\tau = \langle n, q, y\rangle$, respectively. If $\mathtt{tp}(x) \neq q$, then case (ii) holds. If $\mathtt{tp}(x) = q$, then the combined effect of the commands is the same as that of the command $\omega = \langle n, t, y\rangle$. In general only $\sqsubseteq$ is true as $\sigma = \langle n, t, x\rangle$ could break a filesystem on which $\omega = \langle n, t, y\rangle$ works. □

To maintain the tree property, certain command pairs on successive nodes can only be executed in a certain order. This notion is captured by the binary relation $\sigma \ll \tau$.

Definition. For a command pair σ, $\tau \in \Omega$ the binary relation $\sigma \ll \tau$ holds if the pair matches either $\langle n, \mathbb{DF}, \mathbb{O}\rangle \ll \langle{\uparrow}n, \mathbb{D}, \mathbb{FO}\rangle$ or $\langle{\uparrow}n, \mathbb{OF}, \mathbb{D}\rangle \ll \langle n, \mathbb{O}, \mathbb{FD}\rangle$. ◀

Observe that $\sigma \ll \tau$ implies that σ and τ are structural commands on consecutive nodes, and either both are up commands, or both are down commands. Also, if $\sigma_1 \ll \sigma_2 \ll \sigma_3$ then all three commands are in the same category, thus the corresponding nodes are going up or going down. In particular, there are no commands which would form a $\ll$-cycle.

Definition. Let σ be on node n, and τ be on a different node m. The (symmetric) binary relation $\sigma \parallel \tau$ holds in the following cases: n and m are uncomparable; or if n and m are comparable then either the command on the higher node matches $\langle\cdot, \mathbb{D}, \mathbb{D}\rangle$, or the command on the lower node matches $\langle\cdot, \mathbb{O}, \mathbb{O}\rangle$, or both. ◀

Proposition 5 (Command pairs 2)**.** *Suppose σ and τ are on different nodes.*
(a) *$\sigma \parallel \tau$ if and only if $\sigma\tau \equiv \tau\sigma \not\equiv \epsilon$;*
(b) *if $\sigma \nparallel \tau$ then $\sigma\tau \not\equiv \epsilon \Leftrightarrow \sigma \ll \tau$.*

Proof. Tedious, but straightforward case by case checking. □

An immediate consequence of Proposition 5 is

Proposition 6. *If σ and τ are on different nodes, then exactly one of the following three possibilities hold: $\sigma \ll \tau$, or $\tau \ll \sigma$, or $\sigma\tau \equiv \tau\sigma$.* □

A. Rewriting rules

Statements in Propositions 4 and 5 can be considered as *rewriting rules* on command sequences where the command pair $\sigma\tau$ on the left hand side can be replaced by one or two commands on the right hand side. Let us summarize these rewriting rules for future use.

Proposition 7 (Rewriting rules)**.** *For a command pair $\sigma\tau$,*
(a) *if σ, τ are on the same node, then either $\sigma\tau \equiv \epsilon$, or $\sigma\tau \sqsubseteq \omega$ for some $\omega \in \Omega$ which is on the same node as σ and τ are;*
(b) *if σ, τ are on different nodes and $\sigma \not\ll \tau$, then $\sigma\tau \equiv \tau\sigma$ if $\sigma \parallel \tau$, and $\sigma\tau \equiv \epsilon$ otherwise.* □

The syntactical rewriting rules indicated in Proposition 7 fall into three patterns:

$\sigma\tau \equiv \tau\sigma$ (σ and τ commute);
$\sigma\tau \equiv \epsilon$ (the pair, in this order, breaks every filesystem);
$\sigma\tau \sqsubseteq \omega$ for some single command ω.

In the first two cases the rule preserves semantics, while in the last case extends it. To handle breaking sequences seamlessly, three additional rules are added expressing that ϵ is an absorbing element [3]:

$$\epsilon\epsilon \equiv \epsilon, \quad \sigma\epsilon \equiv \epsilon, \quad \epsilon\sigma \equiv \epsilon.$$

Definition. For two command sequences $\alpha \overset{\mathsf{w}}{\sqsubseteq} \beta$ denotes that there is a rewriting sequence using the above rules which produces β from α; and $\alpha \overset{\mathsf{w}}{\equiv} \beta$ denotes that there is a rewriting sequence using semantic preserving rules only. ◀

Observe that $\alpha \overset{\mathsf{w}}{\equiv} \beta$ is not symmetric but clearly transitive. With an abuse of notation, we write $\alpha \overset{\mathsf{w}}{\equiv} \beta$ to mean that either both $\alpha \overset{\mathsf{w}}{\equiv} \epsilon$ and $\beta \overset{\mathsf{w}}{\equiv} \epsilon$, or $\alpha \overset{\mathsf{w}}{\equiv} \beta$ which clearly makes $\overset{\mathsf{w}}{\equiv}$ symmetric. This extended notation lets us rephrase Proposition 6 in terms of rewriting:

Proposition 8. *If σ and τ are on different nodes, then exactly one of the following three possibilities hold: $\sigma \ll \tau$, $\tau \ll \sigma$, or $\sigma\tau \overset{\mathsf{w}}{\equiv} \tau\sigma$.* □

B. Functional completeness

The command set Ω is sufficiently rich to allow transforming any filesystem into any other one assuming that they differ at finitely many nodes only. The proof is constructive, meaning that it not only proves the existence of, but actually specifies an algorithm that creates, such a sequence.

Theorem 9. *The command set Ω is complete in the following sense. Let Φ_0 and Φ_1 be two filesystems differing at finitely many nodes only. There is a command sequence $\alpha \in \Omega^*$ which transforms the first filesystem to the other one as $\alpha\Phi_0 = \Phi_1$.*

Proof. By induction on the number of nodes Φ_0 and Φ_1 differ. If this number is zero, let α be λ. Otherwise let n be one of the lowest nodes where $\Phi_0(n) = x_0$ and $\Phi_1(n) = x_1$ differ, that is, where $\Phi_0(m) = \Phi_1(m)$ for every node m below n.

Let Φ_0' be $\Phi_{0[n\to x_1]}$ and Φ_1' be $\Phi_{1[n\to x_0]}$. Clearly, the number of nodes at which Φ_0' and Φ_1 differ is one less. If Φ_0' is not broken, then the induction gives α' for which $\alpha'\Phi_0' = \Phi_1$, and we set α to $\langle n, \mathsf{tp}(x_0), x_1\rangle\alpha'$. This does not break the filesystem because $\Phi_0' = \langle n, \mathsf{tp}(x_0), x_1\rangle\Phi_0$. Similarly, if Φ_1' is not broken, then the induction gives $\alpha'\Phi_0 = \Phi_1'$ for some α', and we set α to $\alpha'\langle n, \mathsf{tp}(x_0), x_1\rangle$.

It remains to show that either $\Phi_0' \in \mathcal{X}$ or $\Phi_1' \in \mathcal{X}$, which holds if the corresponding function has the tree property. It is trivial if x_0 and x_1 have the same data type. Otherwise, as Φ_0 and Φ_1 have the same values below n, the filesystem in which the value at n is downgraded (in the sense that $\mathbb{D} > \mathbb{F} > \mathbb{O}$) will retain the tree property. $\square$

Actually, a stronger statement has been proved. The command sequence transforming Φ_0 to Φ_1 consists of the commands

$$\{\langle n, \mathsf{tp}(\Phi_0(n)), \Phi_1(n)\rangle \, : n \in \mathbb{N} \text{ and } \Phi_0(n) \neq \Phi_1(n)\}$$

in some order. In particular, each command in α is on a different node.

IV. Simple sequences

Since rewriting rules are semantically correct, $\alpha \overset{\mathsf{w}}{\equiv} \beta$ implies $\alpha \equiv \beta$, and $\alpha \overset{\mathsf{w}}{\sqsubseteq} \beta$ implies $\alpha \sqsubseteq \beta$. The natural question arises whether this set of rewriting rules is *complete*, meaning that the converse implication also holds: $\alpha \equiv \beta$ implies $\alpha \overset{\mathsf{w}}{\equiv} \beta$. We will prove in Theorem 18 that this is indeed the case for the special class of *simple sequences*.

Definition. A finite command sequence is *simple* if it contains at most one command on each node. ◀

Simple sequences form a semantically rich subset of Ω^*: every command sequence can be turned into a simple one while extending its semantics (Theorem 12). At the same time non-breaking simple sequences have strong structural properties (Theorem 14), which makes them suitable for reconciliation.

Definition. (a) The simple sequence $\alpha \in \Omega^*$ *honors* $\ll$ provided that if two commands σ and τ from α satisfy $\sigma \ll \tau$, then σ precedes τ in α.
(b) The command $\sigma \in \alpha$ is a *leader in* α if it is $\ll$-minimal, that is, there is no $\tau \in \alpha$ for which $\tau \ll \sigma$. In particular, each transient command is a leader. ◀

Lemma 10. (a) *Suppose for a simple sequence $\alpha = \alpha_1\sigma\alpha_2$ where σ is a leader in α. If $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$, then $\alpha \overset{\mathsf{w}}{\equiv} \sigma\alpha_1\alpha_2$.*
(b) *If the simple sequence α does not honor $\ll$, then $\alpha \overset{\mathsf{w}}{\equiv} \epsilon$.*
(c) *Let β be a permutation of the simple sequence $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$ such that β honors $\ll$. Then $\alpha \overset{\mathsf{w}}{\equiv} \beta$.*

Proof. All statements are consequences of the facts that exactly one of $\sigma\tau \overset{\mathsf{w}}{\equiv} \tau\sigma$, $\sigma \ll \tau$ and $\tau \ll \sigma$ holds (Proposition 8), together with $\tau\sigma \overset{\mathsf{w}}{\equiv} \epsilon$ when $\sigma \nparallel \tau$ and $\tau \not\ll \sigma$ (Proposition 5).

(a) Let τ be the last command in α_1. Then $\tau\sigma \overset{\mathsf{w}}{\equiv} \sigma\tau$ as we cannot have neither $\tau \ll \sigma$ (as σ is a leader), nor $\sigma \ll \tau$ (as $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$).

(b) Let $\sigma \ll \tau$ and consider the simple sequence $\tau\alpha\sigma$. Let the last command in α be σ_1. If it commutes with σ, then swap them, and continue investigating the remaining commands between τ and σ. If they do not, $\sigma_1 \ll \sigma$ must hold as otherwise $\tau\alpha\sigma$ would rewrite to ϵ. Then, if σ_1 commutes with σ_2, the command before it, swap them as before, otherwise $\sigma_2 \ll \sigma_1$. Ultimately the command we get following τ is σ_k for some k. Now τ and σ_k must be on comparable nodes; both of them are structural commands (thus $\tau \nparallel \sigma_k$), and $\tau \not\ll \sigma_k$ as there are no $\ll$-cycles, therefore $\tau\sigma_k \overset{\mathsf{w}}{\equiv} \epsilon$.

(c) By induction on the length of α. Let σ be a leader in α, then $\alpha \overset{\mathsf{w}}{\equiv} \sigma\alpha_1$ by (a). Let $\beta = \beta_1\sigma\beta_2$. We claim $\tau\sigma \overset{\mathsf{w}}{\equiv} \sigma\tau$ for all $\tau \in \beta_1$. It is so as $\tau \not\ll \sigma$ (σ is a leader), and $\sigma \not\ll \tau$ (since β honors $\ll$). Thus $\beta \overset{\mathsf{w}}{\equiv} \sigma\beta_1\beta_2$, and the induction on α_1 and $\beta_1\beta_2$ gives the claim. $\square$

Lemma 11. *Let $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$ be a simple sequence which contains the commands σ and τ on (comparable) nodes n and m such that $\sigma \nparallel \tau$. Then σ and τ are structural commands, and α contains structural commands on all nodes between n and m.*

Proof. If n and m are immediately related, the claim is a consequence of Proposition 8.

If not, let α be the shortest counterexample to this claim. Then α must start with σ and end with τ (we can assume this order), as otherwise a shorter counterexample would exist. Also, α must contain more than two commands, as by Proposition 5 it would otherwise rewrite to ϵ.

Of σ and τ, we consider the command that is on the lower node. If it is σ, isolate the first two commands in $\alpha = \sigma\sigma'\beta$. If $\sigma\sigma' \overset{\mathsf{w}}{\equiv} \sigma'\sigma$, then $\sigma\beta$ would be a shorter counterexample. Consequently, by Proposition 8, $\sigma \ll \sigma'$. It means that σ is a structural command, and σ' is a structural command on an immediate relative of n which is still below m. Therefore $\sigma' \nparallel \tau$, and $\sigma'\beta$ would be be a shorter counterexample. If τ is on the lower node, we isolate the last two commands in $\alpha = \beta\tau'\tau$ and proceed in a similar fashion. $\square$

Theorem 12 (Rewriting theorem)**.** *For each command sequence α either $\alpha \overset{\mathsf{w}}{\equiv} \epsilon$ or there is a simple sequence α^* such that $\alpha \overset{\mathsf{w}}{\sqsubseteq} \alpha^*$.*

Proof. We assume α is not simple. Let σ be the first command in α for which there is an earlier command τ on the same node. Let this node be n. Splitting α around these commands we get

$$\alpha = \beta\tau\gamma\sigma\beta'.$$

Now $\tau\gamma$ and $\gamma\sigma$ are simple sequences. We claim that $\tau\gamma\sigma$ simplifies (or rewrites to ϵ), which is proved by induction on the length of γ. If γ is empty, then it is guaranteed by Proposition 4. Otherwise let τ' be the first command in γ. If $\tau\tau' \overset{\mathsf{w}}{\equiv} \tau'\tau$, then the induction hypothesis gives the claim. Thus we must have $\tau \ll \tau'$ (as otherwise $\tau\tau'$ would rewrite to ϵ). Similarly, if the last command in γ is σ', then either $\sigma'\sigma \overset{\mathsf{w}}{\equiv} \sigma\sigma'$, when we are done, or $\sigma' \ll \sigma$.

If the length of γ is at least two, then τ' and σ' are on different nodes (as γ is simple). We also know they are on nodes that are immediately related to n, and consequently n is between them. As both τ' and σ' are structural commands, from Lemma 11 we know that γ contains a command on n, which is impossible.

Finally, if γ has a length of one, then $\tau \ll \tau' = \sigma' \ll \sigma$, contradicting the assumption that τ and σ are on the same node. □

Definition. (a) A *$\ll$-chain* is a command sequence $\sigma_1 \ll \sigma_2 \ll \cdots \ll \sigma_k$ connecting σ_1 and σ_k (or σ_k and σ_1).
(b) A finite set T of nodes is a *subtree rooted at* $n \in T$ if every other element of T is below n, and if $t \in T$, then nodes between t and n are also in T.
(c) Finally, $S \subset \Omega$ is a *simple set* if all commands in S are on different nodes, and for any two $\sigma, \tau \in S$ either $\sigma \parallel \tau$, or S contains a $\ll$-chain connecting σ and τ. ◀

Some structural properties of simple sets are summarized below to paint an intuitive picture of their structure. To this end let $S \subset \Omega$ be a fixed simple set. Split the set of nodes of the commands in S into three disjoint parts $N_{\mathbb{D}} \cup N_{\mathbb{O}} \cup N_*$ as follows. If a command in S matches $\langle n, \mathbb{D}, \mathbb{D}\rangle$, then put its node n into $N_{\mathbb{D}}$; if it matches $\langle n, \mathbb{O}, \mathbb{O}\rangle$, then put n into $N_{\mathbb{O}}$, otherwise put it into N_*. The following statements are immediate from the definition.

Proposition 13. (a) *No node in $N_{\mathbb{D}}$ is below any node in $N_{\mathbb{O}} \cup N_*$.*
(b) *No node in $N_{\mathbb{O}}$ is above any node in $N_{\mathbb{D}} \cup N_*$.*
(c) *N_* is a disjoint union of subtrees whose roots are pairwise uncomparable.*
(d) *If $\sigma, \tau \in S$, $\sigma \ll \tau$, then σ and τ are on consecutive nodes in the same subtree of N_*. Conversely, if $T \subset N_*$ is one of the subtrees and σ, $\tau \in S$ are commands on consecutive nodes of T, then $\sigma \ll \tau$ or $\tau \ll \sigma$. Moreover, the commands with nodes in T are either all up commands, or all down commands.*
(e) *Leaders ($\ll$-minimal elements) of S are the commands on nodes in $N_{\mathbb{D}} \cup N_{\mathbb{O}}$, on the root nodes of up-subtrees, and on the leaves of down-subtrees.* □

Theorem 14 (Structural theorem of simple sequences)**.** *Let $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$ be a simple sequence. Then*
(a) *α honors the relation $\ll$;*
(b) *if β is a permutation of α and β honors $\ll$, then $\beta \overset{\mathsf{w}}{\equiv} \alpha$;*
(c) *the set of commands in α is a simple set.*
(d) *Suppose the commands of the sequence β form a simple set, and β honors $\ll$. Then $\beta \overset{\mathsf{w}}{\not\equiv} \epsilon$.*

Proof. (a) and (b) has been proved in Lemma 10.

(c) is a direct consequence of Lemma 11. We know the commands in α apply to different nodes, and that for any $\sigma \not\parallel \tau$ there is a chain of commands on immediately related nodes leading from one to the other. The details of the proof show that they must also form a $\ll$-chain.

(d) All applicable rewriting rules are of the form $\sigma\tau \overset{\mathsf{w}}{\equiv} \tau\sigma$ and $\sigma\tau \overset{\mathsf{w}}{\equiv} \epsilon$. It means that $\beta \overset{\mathsf{w}}{\equiv} \epsilon$ if and only if one can use the commutativity rules to rearrange β to contain two consecutive commands to which the $\sigma\tau \overset{\mathsf{w}}{\equiv} \epsilon$ rule would apply. Assume this is the case. Then $\sigma \not\parallel \tau$, thus σ and τ are connected by some $\ll$-chain $\sigma = \sigma_1 \ll \cdots \ll \sigma_k = \tau$. We also know $k > 2$. Observe that $\sigma_1, \ldots, \sigma_k$ must appear in β in this order, and this order remains after applying any commutativity rule. This is a contradiction as then σ and τ can never become consecutive commands. □

Definition. For sequences α, β we write $\alpha \parallel \tau$ to mean $\sigma \parallel \tau$ for all $\sigma \in \alpha$; and write $\alpha \parallel \beta$ to mean $\alpha \parallel \tau$ for all $\tau \in \beta$. ◀

An immediate consequence of Theorem 14 is the following.

Proposition 15. *Let $\alpha\tau$ be a simple sequence where $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$. Then*
(a) *$\alpha\tau \overset{\mathsf{w}}{\not\equiv} \epsilon$ if and only if either $\alpha \parallel \tau$ or $\sigma \ll \tau$ for some $\sigma \in \alpha$.*
(b) *$\tau\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$ if and only if either $\tau \parallel \alpha$ or $\tau \ll \sigma$ for some $\sigma \in \alpha$.* □

Our next goal is to show that on simple sequences the set of rewriting rules is semantically complete in a strong sense. To this end we first state a result which shows that the filesystem commands capture a surprising amount of information. If the simple sequence α does not break Φ, then clearly Φ must match the input type of every command in α. This simple necessary condition is almost sufficient.

Theorem 16. *Let $\alpha \overset{\mathsf{w}}{\not\equiv} \epsilon$ be a simple sequence and $\Phi \in \mathcal{X}$ be a filesystem. α does not break Φ if and only if the following conditions hold for each $\sigma \in \alpha$:*
(a) *If σ is on node n, then $\Phi(n)$ has the data type required by σ.*
(b) *If σ is a leader matching $\langle n, \mathbb{O}, \mathbb{FD}\rangle$, then the nodes above n are directories.*
(c) *If σ matches $\langle n, \mathbb{D}, \mathbb{FO}\rangle$, then nodes below n not mentioned in α are empty.*

Proof. The conditions are clearly necessary. For the converse use induction on the length of α. Let Φ be a filesystem satisfying the conditions for commands in $\tau\alpha$, where τ is on node m. Clearly, τ can be applied to Φ as τ is a leader, thus it suffices to check that $\tau\Phi$ satisfies the conditions for $\sigma \in \alpha$. Let σ be on node n. (a) clearly holds as m and n are different. For (b) observe that by Proposition 15 if σ is a leader in α, then either it is a leader in $\tau\alpha$ (and then $\tau \parallel \sigma$) or $\tau \ll \sigma$. In the first case either m and n are incomparable, or m is below n, or τ matches $\langle n, \mathbb{D}, \mathbb{D}\rangle$. In all cases $\tau\Phi$ and Φ have the same types of values above n. We know σ is an up command, so in the second case τ is one, too, and it is on the parent node of that of σ. $\tau\Phi[m]$ is a directory, and thus every other node above it is a directory as well.

The reason why (c) holds is similar. If $\tau \parallel \sigma$, then either m is above n, or below it and τ matches $\langle m, \mathbb{O}, \mathbb{O}\rangle$. Otherwise there is a $\ll$-chain connecting τ and σ consisting of down commands only. Thus τ matches $\langle m, \mathbb{DF}, \mathbb{O}\rangle$, and therefore $\tau\Phi[m]$ is empty. □

The following corollary, which merges the last two conditions, will be used when constructing non-breaking filesystems.

Corollary 17. *Let α be a simple sequence, and let Φ_1 and Φ_2 be filesystems so that $\Phi_1(n)$ and $\Phi_2(n)$ have the same types for every node n which is* (a) *the node of some command in α, and* (b) *comparable to the node of a structural command in α. Then $\alpha\Phi_1 \neq \perp$ iff $\alpha\Phi_2 \neq \perp$.* □

Theorem 18 (Completeness theorem for simple sequences)**.** *For a simple sequence α, if $\alpha \equiv \epsilon$, then $\alpha \stackrel{\mathsf{w}}{\equiv} \epsilon$. In general, if α and β are simple sequences such that $\alpha \equiv \beta$, then $\alpha \stackrel{\mathsf{w}}{\equiv} \beta$ provided there are at least two different values in each data type.*

Proof. a) Assume $\alpha \stackrel{\mathsf{w}}{\not\equiv} \epsilon$. It suffices to construct a filesystem Φ which satisfies the conditions of Theorem 16. Start with the empty filesystem. For each leader $\sigma \in \alpha$ on a node n that does not match $\langle n, \mathbb{O}, \mathbb{O}\rangle$, change the nodes above n in Φ to a directory value, and set n to a value matching the input type of σ.

After the process finishes, condition (a) of Theorem 16 clearly holds for leaders. If τ is not a leader, then it is either an up command or a down command. In the first case its leader is above τ, thus the node of τ is empty in Φ (leaders are on uncompatible nodes), as required. If τ is a down command, then its leader is below τ, and then the node of τ is a directory node. Condition (b) is clear from the construction. For (c) remark that non-empty values are only above nodes mentioned in α.

b) Assume α and β are simple sequences, and none of them breaks every filesystem. If they contain the same command set, then (b) of Theorem 14 gives the claim, thus it suffices to show that, e.g., each command in β is also in α. Let $\sigma \in \beta$ be a command on node n, and Φ be a filesystem on which both α and β work. If α does not contain any command on n, then $(\beta\Phi)(n)$ differs from $\Phi(n) = (\alpha\Phi)(n)$, contradicting $\alpha \equiv \beta$, except when the replacement value in σ is the same as $\Phi(n)$. In this case, however, $\Phi(n)$ can be replaced by another value from the same data type to force $\Phi(n)$ and $(\sigma\Phi)(n)$ be different. If α contains a command τ on n, then the input types of σ and τ must be the same, and the replacement values of σ and τ must be the same, thus σ and τ are the same commands. $\square$

In the case when there is only one value in some data type, the second part of the theorem does not remain true. For example, if there is only one element in the empty data type, then $\langle n, \mathbb{O}, \mathsf{o}\rangle$ commands are guaranteed not to make any change to the filesystem, thus α and β may contain additional commands of this type without changing their semantics. If these “no-operation” commands are deleted from the equivalent α and β sequences extending their semantics, or if the rewriting rules are extended with removal rules for such commands as in [5], then they become rewritable.

V. Update detection

Given the original filesystem Φ and its copy Φ_1 modified at finitely many nodes, determine the simple update sequence $\alpha \in \Omega^*$ for which $\alpha\Phi = \Phi_1$ as follows:

1) For each node $n \in \mathbb{N}$ where $\Phi(n)$ and $\Phi_1(n)$ differ, add the command $\langle n, \mathtt{tp}(\Phi(n)), \Phi_1(n)\rangle$ to the command set S.
2) Order S to become sequence α which honors $\ll$.

The set S can be created by traversing Φ and Φ_1 simultaneously. Every command set can be ordered to honor $\ll$ by first creating the transitive closure of $\ll$, and then using topological sort. The sorting procedure is clearly quadratic in the number of commands.

Theorem 19 (Correctness of update detection)**.** *The simple command sequence $\alpha \in \Omega^*$ returned by the update detector works as expected: $\alpha\Phi = \Phi_1$.*

Proof. By Theorem 9 there is a command sequence β such that $\beta\Phi = \Phi_1$, and this command set consists of exactly the commands in the above set S. By Theorem 14 this set S is simple, and any ordering of S honoring $\ll$ is semantically equivalent to β. $\square$

Theorem 20. *Suppose $\Phi_1 = \alpha^*\Phi$ for some command sequence α^*. Then there is a simple sequence α such that $\Phi_1 = \alpha\Phi$. This α can be computed from α^* in quadratic time.*

Proof. By Theorem 12 there exists a simple sequence α such that $\alpha^* \stackrel{\mathsf{w}}{\sqsubseteq} \alpha$, and then $\alpha\Phi = \Phi_1$. The proof also indicates a quadratic algorithm generating α. For each command $\sigma \in \alpha^*$ search backward from σ to find the first command $\tau \in \alpha^*$ which is on the same node as σ. If such a τ is found, then use commutativity rules to move τ ahead and σ backward until τ and σ are next to each other, and then replace them by a single command. $\square$

VI. Refluent sequences

Recall that two simple command sequences α and β are *refluent* if there is a filesystem which neither α nor β breaks. Using the $\vDash$ notation, it can be expressed as $\{\alpha, \beta\} \nvDash \epsilon$. This section starts with a characterization of refluent pairs in the special case when the node sets of α and β are disjoint. A general reduction theorem together with a partial converse is provided for the case when α and β share commands on the same node.

Theorem 21 (Refluent sequences)**.** *The node-disjoint non-breaking simple sequences α and β are refluent if and only if for each leader σ in α one of the following conditions hold:*

- *$\sigma \parallel \beta$, or*
- *σ matches $\langle n, \mathbb{O}, \mathbb{FD}\rangle$ and there is a command on $\uparrow n$ in β matching $\langle \uparrow n, \mathbb{D}, \mathbb{FO}\rangle$, or*
- *σ matches $\langle \uparrow n, \mathbb{D}, \mathbb{FO}\rangle$ and some leader in β matches $\langle n, \mathbb{O}, \mathbb{FD}\rangle$;*

and the symmetric statements hold for each leader in β.

Proof. Assume first that α, β satisfy the above conditions. Create the filesystem Φ by repeating the process indicated in Theorem 18 for both α and β: start from the empty filesystem, and for each leader in α and in β execute the described modifications of Φ. Conditions of this theorem guarantee that Theorem 16 applies equally to Φ and α and to Φ and β.

For the converse suppose α and β are refluent and σ is a leader in α. By Theorem 14 there is an equivalent permutation of α which starts with σ; and then the pair $\{\sigma, \beta\}$ is refluent as well. Thus we may suppose that the first sequence consists of this single leader only; let the node of σ be n.

If $\sigma \| \beta$ then we are done, so suppose otherwise, which means that there is a command $\tau \in \beta$ on a node m comparable to n.

Let Φ be a filesystem neither σ nor β breaks. If m is below n, then $\Phi(n)$ is a directory (as τ is not an $\langle n,\mathbb{O},\mathbb{O}\rangle$ command and there are no commands on n in β). As σ changes $\Phi(n)$ to a non-directory, every node below n is empty – in particular, $\Phi(m)$ is empty. Now τ changes $\Phi(m)$ to a non-empty value, thus τ is an up command. Consider the leader of the subtree of τ, let it be τ' on node $m'\preceq m$. There are no commands in β above m', and there are structural commands in β on every node between m' and m. Consequently m' is below n, and just before τ' is executed, the content of all nodes on and above m' are the same as in Φ. As in Φ all nodes below n are empty, m' must be the child of n, and the leader τ' matches $\langle m',\mathbb{O},\mathbb{FD}\rangle$, while σ matches $\langle{\uparrow}m',\mathbb{D},\mathbb{FO}\rangle$.

The last case is when n is below the node of some command in β. Consider $\tau\in\beta$ on the node m above n such that no command in β is on a node between n and m. As σ is not a $\langle n,\mathbb{O},\mathbb{O}\rangle$ command, and $\sigma\Phi$ is not broken, all nodes above m are directories. τ is not $\langle m,\mathbb{D},\mathbb{D}\rangle$, and when executed, the content at nodes between n and m is the original value of Φ. As β does not break Φ, m must be the parent of n, and $\Phi(n)$ must be empty, leading to the second possibility. ☐

Next we consider the general case when the simple sequences can share commands on the same node. The following lemmas will be used later.

Lemma 22. *Suppose α, β are refluent simple sequences and τ_1, $\tau_2\in\beta$ are such that $\tau_1\ll\tau_2$. If $\tau_2\parallel\alpha$, then $\tau_1\parallel\alpha$.*

Proof. Consider first the case when $\tau_1\ll\tau_2$ matches $\langle n,\mathbb{DF},\mathbb{O}\rangle\ll\langle{\uparrow}n,\mathbb{D},\mathbb{FO}\rangle$. It suffices to consider commands $\sigma\in\alpha$ which are on a node comparable to n. If there is a σ on n, then its input type is the same as that of τ_1, namely not $\mathbb{O}$, and then $\tau_2\not\parallel\sigma$. For the same reason σ cannot be on ${\uparrow}n$. Thus σ is either below n, or above ${\uparrow}n$, and in both cases $\tau_2\parallel\sigma$ implies $\tau_1\parallel\sigma$.

In the second case the pair matches $\langle{\uparrow}n,\mathbb{OF},\mathbb{D}\rangle\ll\langle n,\mathbb{O},\mathbb{FD}\rangle$. $\sigma\in\alpha$ cannot be on ${\uparrow}n$ as the input type of τ_1 is not $\mathbb{D}$, and so $\tau_2\not\parallel\sigma$ would hold. Otherwise, if σ is above ${\uparrow}n$ then $\tau_2\parallel\sigma$ implies $\tau_1\parallel\sigma$. In the remaining cases the node m of σ is below ${\uparrow}n$. The input type of τ_1 is not a directory, thus every command below ${\uparrow}n$ must have $\mathbb{O}$ as input type by Theorem 16. If the output type of σ is also $\mathbb{O}$, then we are done. If not, then σ is an up command, and consider the leader (in α) of σ; there is a command in α on every node between m and the leader. Every node above the leader is a directory (Theorem 16), thus this leader must be on or above ${\uparrow}n$. But then α has a command on ${\uparrow}n$, which is a contradiction. ☐

Lemma 23. *Suppose $\tau\alpha$ and β are non-breaking simple sequences, and $\tau\parallel\beta$. If α and β are refluent, then so are $\tau\alpha$ and β.*

Proof. Let τ be on the node n, and Φ be a filesystem which neither α nor β breaks. Our aim is to construct a filesystem Ψ on which both $\tau\alpha$ and β work. As $\tau\alpha$ is non-breaking, by Proposition 15 either $\tau\parallel\alpha$ or $\tau\ll\sigma$ for some $\sigma\in\alpha$.

If $\tau\parallel\alpha$, then to get Ψ, change $\Phi(n)$ to a value matching the input type of τ, all nodes above n to a directory value (except when τ matches $\langle n,\mathbb{O},\mathbb{O}\rangle$), and all nodes below n to an empty value (except when τ matches $\langle n,\mathbb{D},\mathbb{D}\rangle$). Clearly, τ is applicable to Ψ, and according to Corollary 17, neither β breaks Ψ, nor α breaks $\tau\Psi$.

If $\tau\ll\sigma$ matches $\langle n,\mathbb{DF},\mathbb{O}\rangle\ll\langle{\uparrow}n,\mathbb{D},\mathbb{FO}\rangle$, then $\Phi({\uparrow}n)$ is a directory, and $\Phi(n)$ is empty by Theorem 16 (as n is not mentioned in α). To get Ψ, change Φ only at n to a value matching the input type of τ. Then Ψ is a filesystem, and due to Corollary 17 β does not break Ψ, and α does not break $\tau\Psi$.

Finally, if τ matches $\langle n,\mathbb{OF},\mathbb{D}\rangle$, then σ is an up command on a child of n, consequently $\Phi(n)$ is a directory. All commands in α and in β with a node below n have the empty input type. In this case in Φ change the content of every node below n to an empty value, and at n to a value matching the input type of τ. Then Ψ is a filesystem, and, as before, both $\tau\alpha$ and β can be applied to Ψ. ☐

Recall that $\mathsf{tp}(\sigma)=\mathsf{tp}(\tau)$ if these commands are on the same node, and have the same input and the same output types.

Lemma 24. *Suppose α and β are refluent simple sequences, $\sigma\in\alpha$, $\tau\in\beta$, and $\sigma\ll\tau$. Then there is a $\sigma'\in\beta$ such that $\mathsf{tp}(\sigma)=\mathsf{tp}(\sigma')$.*

Proof. Let Φ be a filesystem on which both α and β work, and suppose by contradiction that β has no structural command on the node of σ. Distinguish two cases based on the pattern of $\sigma\ll\tau$. If it is $\langle n,\mathbb{DF},\mathbb{O}\rangle\ll\langle{\uparrow}n,\mathbb{D},\mathbb{FO}\rangle$, then $\Phi(n)$ matches the input type of σ (as otherwise α would break Φ), in particular $\Phi(n)$ is not empty. As β has no structural command on n, β does not change the type of $\Phi(n)$. But when τ is executed, $\Phi({\uparrow}n)$ becomes a non-directory, which breaks the filesystem.

In the second case the pattern is $\langle{\uparrow}n,\mathbb{OF},\mathbb{D}\rangle\ll\langle n,\mathbb{O},\mathbb{FD}\rangle$. Then $\Phi({\uparrow}n)$ is not a directory and β keeps the value type stored here. However τ adds a non-empty value at n breaking the filesystem.

Thus β contains a structural command σ' on the node of σ. Since α and β are refluent simple sequences, the input types of σ and σ' are the same (both are applicable to the same node). By Theorem 14 $\sigma'\ll\tau$ must also hold (as σ' and τ are structural commands on immediate relatives in a non-breaking simple sequence). Combined with the fact that $\sigma\ll\tau$ and $\sigma'\ll\tau$ implies that σ and σ' have the same output type, the lemma follows. ☐

Definition. Write $\alpha\cap^{\mathsf{tp}}\beta$ for the set of commands from α which are on the same node, have the same input type and the same output type as some command in β:

$$\alpha\cap^{\mathsf{tp}}\beta=\{\sigma\in\alpha:\mathsf{tp}(\sigma)=\mathsf{tp}(\tau)\text{ for some }\tau\in\beta\}.$$

◀

Clearly, if α and β are simple, the elements of $\alpha\cap^{\mathsf{tp}}\beta$ and $\beta\cap^{\mathsf{tp}}\alpha$ are in a one-to-one correspondence: a command in one set corresponds to the command in the other set on the same node; the pairs share the node, the input type, and the output type.

Theorem 25 (Reduction of refluent sequences)**.** *Suppose α and β are refluent simple sequences. Then $\alpha\equiv\alpha_1\alpha_2$ and*

$\beta \equiv \beta_1\beta_2$ where α_1 consists of commands in $\alpha \cap^{\mathsf{tp}} \beta$, and β_1 consists of commands in $\beta \cap^{\mathsf{tp}} \alpha$. Furthermore α_2 and β_2 are refluent.

Proof. The first part of the theorem follows from the structural Theorem 14 after we show that if for any two commands $\sigma \ll \tau$ from α such that $\tau \in \alpha_1$, then $\sigma \in \alpha_1$ as well. But this is immediate from Lemma 24.

To see the second part, observe that the simple command sets of α_1 and β_1 are on the same node set with the same input and output types. Consequently for any filesystem Φ, $\alpha_1\Phi$ and $\beta_1\Phi$ have the same data type (but not necessarily the same value) at every node. Thus if $\alpha_2(\alpha_1\Phi) \neq \perp$ and $\beta_2(\beta_1\Phi) \neq \perp$, then $\beta_2(\alpha_1\Phi) \neq \perp$ as well, showing that α_2 and β_2 are refluent indeed. □

A partial converse of Theorem 25 is true.

Theorem 26. *Suppose $\gamma\alpha$ and $\gamma\beta$ are non-breaking simple sequences. They are refluent if and only if α and β are refluent.*

Proof. The direction that if $\gamma\alpha$ and $\gamma\beta$ are refluent, then so are α and β is clear. The other direction follows from the special case when γ consists of a single command τ. For this case, however, an easy adaptation of the proof of Lemma 23 works. □

VII. Reconciliation

Let us revisit the problem of file synchronization. We have two simple sequences α and β which create two divergent replicas of the same original filesystem. The goal is to find (preferably maximal) subsets which can then be carried over to the other copy without destroying local modifications.

Definition. (a) The sequence β' formed from commands in β is a *reconciler for α over β*, if β' does not destroy any local change made by α, and is always applicable after α in the sense that $\{\alpha, \beta\} \vDash \alpha\beta'$.
(b) The sequences α and β are *confluent* if they are refluent, and there are reconcilers β' and α' which create identical results, written succinctly as $\{\alpha, \beta\} \vDash \alpha\beta' \equiv \beta\alpha'$. ◀

Theorem 27 (Confluent node-disjoint sequences)**.** *The node-disjoint non-breaking simple sequences α and β are confluent if and only if $\alpha \parallel \beta$. In this case the reconciler sequences are β and α respectively as $\{\alpha, \beta\} \vDash \alpha\beta \equiv \beta\alpha$.*

Proof. Suppose α and β are applied to the filesystem Φ. We may assume that each command actually changes the content of the filesystem. Let β' and α' be the reconcilers, that is, $\alpha\beta' \equiv \beta\alpha'$. If $\sigma \in \beta$ on node n were not in β', then $(\alpha\beta')\Phi$ has the original content at n, while $(\beta\alpha')\Phi$ has a different value as changed by σ. Thus β' contains all commands of β, and as it is non-breaking, it is equivalent to β by Theorem 14. Consequently $\beta' = \beta$ and $\alpha' = \alpha$ satisfy $\alpha\beta \equiv \beta\alpha$. As $\alpha\beta$ and $\beta\alpha$ are non-breaking simple sequences, both honor $\ll$. By Theorem 14 it means that $\alpha \parallel \beta$.

For the other direction assume α, β are node-disjoint, non-breaking sequences such that $\alpha \parallel \beta$. By Theorem 14 $\alpha\beta \equiv \beta\alpha$, thus it suffices to show that $\{\alpha, \beta\} \vDash \alpha\beta$. The command sets of α and β are simple. As $\alpha \parallel \beta$, the same is true for the command set of $\alpha \cup \beta$; moreover $\alpha\beta$ is an ordering of this simple set which honors $\ll$. In particular, $\alpha\beta$ is non-breaking. Let Φ be a filesystem which neither α nor β breaks, that is, conditions of Theorem 16 hold for α and β. Since $\alpha \parallel \beta$, every leader in $\alpha\beta$ is either a leader in α, or is a leader in β. From here it follows that the same conditions hold for the sequence $\alpha\beta$, meaning $(\alpha\beta)\Phi \neq \perp$, as required. □

To state the main result of this section we need some additional definitions. Recall that $\alpha \cap^{\mathsf{tp}} \beta$ is the set of those commands from α which have the same node, same input type and same output type as some command in β.

Definition. The set of commands in α not in $\alpha \cap^{\mathsf{tp}} \beta$ is denoted by $\alpha \setminus^{\mathsf{tp}} \beta$ as

$$\alpha \setminus^{\mathsf{tp}} \beta = \{\sigma \in \alpha : \text{for every } \tau \in \beta,\ \ \mathsf{tp}(\sigma) \neq \mathsf{tp}(\tau)\}.$$

Finally, let us define

$$\mathcal{R}(\beta \mid \alpha) = \{\tau \in \beta \setminus^{\mathsf{tp}} \alpha : \tau \parallel \alpha \setminus^{\mathsf{tp}} \beta\}.$$

When this set is used as a sequence, it is ordered so that the ordering honors $\ll$. Any two such ordering gives equivalent sequences by Theorem 14. ◀

Theorem 28. *Let α, β be refluent simple sequences. Then*
(a) *$\mathcal{R}(\beta \mid \alpha)$ is a reconciler for α over β.*
(b) *If β' is a reconciler, then $\beta' \subseteq \mathcal{R}(\beta \mid \alpha)$.*

Proof. (a) By the Reduction Theorem 25, α and β can be equivalently rearranged as $\alpha_1\alpha_2$ and $\beta_1\beta_2$ where α_1 consists of the commands of $\alpha \cap^{\mathsf{tp}} \beta$, α_2 consists of the commands of $\alpha \setminus^{\mathsf{tp}} \beta$, and similarly for β_1 and β_2. Recall that by the same theorem α_2 and β_2 are also refluent.

We claim that β_2 can be rearranged so that it starts with $\mathcal{R}(\beta \mid \alpha)$. To this end we only need to show that if $\tau_1 \ll \tau_2$ are in β_2 and $\tau_2 \in \mathcal{R}(\beta|\alpha)$, then so is τ_1. By definition $\tau \in \beta_2$ is in $\mathcal{R}(\beta \mid \alpha)$ iff $\tau \parallel \alpha_2$. Thus $\tau_2 \parallel \alpha_2$, and then by Lemma 22 we have $\tau_1 \parallel \alpha_2$, as required.

Therefore $\beta_2 \equiv \beta'\beta''$ where β' consists of the commands in $\mathcal{R}(\beta \mid \alpha)$. Now β' and α_2 are node-disjoint non-breaking refluent sequences such that $\beta' \parallel \alpha_2$. Theorem 27 gives that in this case $\{\alpha_2, \beta'\} \vDash \alpha_2\beta'$, and then by Theorem 26 we have $\{\alpha, \beta\} \vDash \alpha\beta'$, proving that $\mathcal{R}(\beta \mid \alpha)$ is indeed a reconciler.

(b) Suppose β' is a reconciler, in particular $\{\alpha, \beta\} \vDash \alpha\beta'$. Then $\alpha\beta'$ and β are also refluent, consequently, by Theorem 25, $\alpha \equiv \alpha_1\beta'\alpha_2$ and $\beta \equiv \beta_1\beta'\beta_2$ where α_1 and β_1 are the commands from $\alpha \cap^{\mathsf{tp}} \beta$ and $\beta \cap^{\mathsf{tp}} \alpha$, respectively. As $\beta'\alpha_2$ is a non-breaking simple sequence, if $\beta' \nparallel \alpha_2$ then according to Theorem 14 there are $\tau \in \beta'$ and $\sigma \in \alpha_2$ such that $\tau \ll \sigma$. But α and β are refluent, $\tau \in \beta$, $\sigma \in \alpha$, and then Lemma 24 gives that there is a $\tau' \in \alpha$ such that $\mathsf{tp}(\tau') = \mathsf{tp}(\tau)$ meaning that $\tau \in \beta_1$, which is impossible. Thus $\beta' \parallel \alpha_2$ and then $\beta' \subseteq \mathcal{R}(\beta \mid \alpha)$. □

It should be clear that $\mathcal{R}(\beta \mid \alpha)$ can be determined from the simple sequences α and β in quadratic time. Split the commands in α into the sets $\alpha \cap^{\mathsf{tp}} \beta$ and $\alpha \setminus^{\mathsf{tp}} \beta$, and similarly for β. Then go over each element of $\beta \setminus^{\mathsf{tp}} \alpha$ and check whether it satisfies $\tau \parallel \alpha \setminus^{\mathsf{tp}} \beta$. As β honors $\ll$, keeping elements of

$\mathcal{R}(\beta\,|\,\alpha)$ in the same order as they are in β provides a correct ordering of $\mathcal{R}(\beta\mid\alpha)$.

We can write the refluent sequences as $\alpha\equiv\alpha_1\mathcal{R}(\alpha\mid\beta)\alpha_3$ where α_1 consists of commands in $\alpha\cap^{\mathsf{tp}}\beta$, and similarly for β. Unresolved conflicts come from two sources. First, the matching commands in α_1 and β_1 might store different values (of the same type) at the same node, which would override a local change made by α. These conflicts should be resolved by some content negotiation. Second, a command $\sigma\in\beta_3$ is either on the same node as some command in α (actually, in α_3) assigning a different value type thus again overriding a local change, or $\sigma\nparallel\alpha_3$. In this latter case executing σ after α (or even after $\alpha\mathcal{R}(\beta\mid\alpha)$) would break the filesystem.

Finally, let us state an immediate consequence of Theorem 28 of which Theorem 27 is a special case.

Theorem 29. *The refluent simple sequences α and β are confluent if and only if $\alpha\cap^{\mathsf{tp}}\beta=\beta\cap^{\mathsf{tp}}\alpha$ and $\alpha\setminus^{\mathsf{tp}}\beta\|\beta\setminus^{\mathsf{tp}}\alpha$.* □

VIII. Conclusion

Our main motivation was to generalize and extend some of the results from [5] in a more abstract setting, and concentrated mainly on proving several characterization results of this intriguing algebraic model. One of the main contributions of [5] is the model of filesystems and filesystem commands, which have been adopted here as well. The semantics of command sequences is defined through their action on the filesystems, which gives rise to semantic equivalence and semantic validity. Two sequences are semantically equivalent if they have the same effect on all filesystems, while the sequence α is semantically valid on a filesystem Φ if α can be executed on Φ without breaking it. This semantical validity shares many properties of the "logically valid" notion of mathematical logic as discussed in Proposition 1. Similarly, the *non-breaking* property of sequences corresponds to that of consistency or satisfiability in logic. The set of filesystem commands is also *functionally complete*: if two filesystems differ at finitely many nodes, then there is a *simple* command sequence transforming one into the other as shown by Theorem 9.

Command sequences can be manipulated syntactically by applying the *rewriting rules* defined in Proposition 7. Every sequence can be rewritten into a simple sequence while extending its semantics (Theorem 12), and two simple sequences are semantically equivalent if and only if they can be rewritten into each other (Theorem 18). By Theorem 14 the semantics of a simple sequence is uniquely determined by the *set of its commands*; each feasible ordering (which can be found in quadratic time) of such a simple set gives a sequence with the same semantics. The existence of an effective update detector algorithm follows easily from these properties yielding the statements in Theorem 2.

Two simple sequences are *refluent* if they are jointly consistent: there is a filesystem which neither of them breaks. The problem of syntactical characterization of such sequence pairs seems to be hard and have been solved only partially. Theorem 21 gives a complete characterization for the special case of node-disjoint sequences. By Theorems 25 and 26 commands on the same node with the same input and same output type can be ignored.

Reconciliation is a relaxed notion of confluence: given two simple sequences α and β, can we add further commands (without overriding the effects of old ones) to these sequences so that $\alpha\beta'\equiv\beta\alpha'$? Theorem 29 gives a complete characterization when sequences are confluent, while by Theorem 28 there is a unique maximal reconciler for any pair of refluent simple sequences. This result justifies Theorem 3. It is an open problem whether they remain valid when the reconciliation should be applied for more than two replicas.

We have assumed that all three data types $\mathbb{D}$, $\mathbb{F}$, $\mathbb{O}$ contain at least two elements; this fact was used in the proof of the Completeness theorem 18. When one (or more) of them has a single element only, the corresponding transient command should be deleted, as explained at the end of Section IV. With that modification all theorems remain valid.

During the preparation of this work we have looked at changing, relaxing, or modifying several parameters of the chosen model, without any success. One such modification was to use more than three value types. The resulting filesystem semantics (using various restrictions on how the values vary along each branch) with the corresponding command set did not lead to any syntactical characterization of semantically equivalent sequences. It is an interesting and intriguing problem to understand why this particular semantics is so powerful and, at the same time, so tractable.

References

[1] F. Baader, T. Nupkow. *Term rewriting and All That*. Cambridge Univ. Press, 1998

[2] S. Balasubramaniam and B. C. Pierce. What is a File Synchronizer? In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking.* New York, NY. ACM, 1998, pp. 98–108

[3] M. Beaudry. Finite Idempotent Grupoids and Regular Languages. *Informatique Thorique et Applications*, 1998, vol 32, pp. 127–140

[4] C. Berzan and N. Ramsey. Summer Scholars Technical Report. *Tufts University*, Medford, MA, 2010. Available as http://thirld.com/files/summerscholars_techreport.pdf

[5] E. P. Csirmaz. Algebraic File Synchronization: Adequacy and Completeness. arXiv:1601.01736

[6] A-M. Kermarrec and A. Rowstron and M. Shapiro and P. Druschel. The IceCube Approach to the Reconciliation of Divergent Replicas. In *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing.* New York, NY. ACM, 2001, pp. 210–218

[7] V. Martins and E. Pacitti and P. Valduriez. Distributed semantic reconciliation of replicated data. in *Proc. of CDUR*, Paris, France. CNAM, 2005, pp. 48–53

[8] B. C. Pierce and J. Vouillon. What's in Unison? A Formal Specification and Reference Implementation of a File Synchronizer. *U. of Pennsylvania Technical Reports (CIS) 40*, 2004. Available as http://repository.upenn.edu/cis_reports/40

[9] N. Ramsey and E. Csirmaz. An Algebraic Approach to File Synchronization. In *Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering.* New York, NY. ACM, 2001, pp. 175–185.

[10] Y. Saito and M. Shapiro. Optimistic Replication. in *ACM Computing Surveys* 2005, vol. 37, no. 1. pp. 42–81.

[11] V. Tao and M. Shapiro and V. Rancurel. Merging Semantics for Conflict Updates in Geo-distributed File Systems. in *Proceedings of the 8th ACM International Systems and Storage Conference.* New York, NY. ACM, 2015, pp. 10:1–10:12.

[12] D. B. Terry et al. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. in *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles.* New York, NY. ACM, 1995, pp. 172–182.