Partial Updates

Yuri Gurevich, Nikolai Tillmann

Microsoft Research, One Microsoft Way, Redmond, WA 98052

Abstract

A datastructure instance, e.g. a set or file or record, may be modified independently by different parts of a computer system. The modifications may be nested. Such hierarchies of modifications need to be efficiently checked for consistency and integrated. This is the problem of partial updates in a nutshell. In our first paper on the subject, we developed an algebraic framework which allowed us to solve the partial update problem for some useful datastructures including counters, sets and maps. These solutions are used for the efficient implementation of concurrent data modifications in the specification language AsmL. The two main contributions of this paper are (i) a more general algebraic framework for partial updates and (ii) a solution of the partial update problem for sequences and labeled ordered trees.

Key words: Abstract state machines; applicative algebra; AsmL; parallel composition; sequences; transactions; updates

1 Introduction

In many modern computer systems, the same file, or file system, or database, etc. can be concurrently modified by many parts of the system. Typically such modifications update only a part of the data. These partial updates of the data need to be checked for consistency and integrated. The phenomenon of partial updates is common. Examples vary from huge airline booking systems to modest counters that record occurrences of parallel events. We came across that phenomenon during the development of the specification language AsmL by the group on Foundations of Software Engineering at Microsoft Research [7]. AsmL is suitable for programming and executing abstract state machines [2,5,9]. Typically, a single step of an abstract state machine involves many child submachines executing in parallel and reporting computation results to their parent machine. The machine and the child submachines modify data stored in various locations. Often submachines are nested; children report to their parents. And each member of this

Email addresses: gurevich@microsoft.com (Yuri Gurevich), nikolait@microsoft.com (Nikolai Tillmann).

hierarchy of machines can interact with outside computer systems within a single step of its own. All this makes AsmL powerful but creates a nontrivial problem of integrating partial updates (and checking them for consistency). We hasten to say that we do not presume any knowledge of abstract state machines in this paper.

We addressed the partial update problem first in [10]. The basic framework there was as follows. Fix a datatype T and consider a monoid of unary operations over T with respect to functional composition; the operations are called particles. The parallel composition of particles is the order-independent functional composition, so that parallel composition is an abstraction of sequential composition when the order of execution is immaterial. The approach was applied to solve the problems of partial updates of counters, sets and maps. Our analysis allowed us to separate the concerns of reporting within the submachine hierarchy from the concern of how to integrate partial updates. Our analysis provided a part of the semantical foundation of AsmL, and our algorithms simplified the implementation of AsmL and made it more efficient.

It turned out, however, that the approach of [10] is too limited. A case in point is that of sequences (which may be called files as well). Consider, for example, two operations inserting elements at different places of a given file. The two insertions are compatible and could be executed simultaneously. However the sequential composition of insertions depends on the order in which they are executed. If the right insertion is executed first then the result coincides with that of the simultaneous execution. But if the left insertion is executed first, it offsets the positions to the right of the inserted element, so that the right insertion will put its element at a wrong place.

Here, in Section 2, we introduce a more general approach to the partial update problem. It is based on the notion of applicative algebra. This new notion was not used in [10] explicitly but special applicative algebras, described in Subsection 2.6, were used there implicitly. We believe that the notion of applicative algebras is of independent interest all by itself.

In Section 3, the new approach is applied to sequences. The sequences could be sequences of characters but we take a more general point of view. We fix an arbitrary applicative algebra A, and we work with sequences of elements of A. The particles of A give rise to certain sequence particles.

The case of sequences is harder than those of counters, sets and maps. This is not so surprising because the problem of partial updates of sequences is related to the difficult collaborative editing problem. Here are a few references on the latter problem. The SCCS [1] and RCS [16] systems manage collections of files which are concurrently edited by different users, but only one user is allowed to edit a file at a time, and thus changes can always be applied sequentially. The CVS system [3] allows overlapping file modifications by several users at a time. All modifications are merged eventually. CVS limits merging to the case where an original file is merged with at most two modified versions of the original file. More specifically, CVS uses the UNIX diff3 [8] command to determine a minimal set of modifications that transform the original file into the modified versions. File modifications are expressed in terms of inserted, deleted, and altered lines. If the modifications do not overlap, they are applied. Otherwise, the user has to resolve the conflict. A different approach is taken by real-time editing systems. They

do not determine file changes *a posteriori*; instead change requests are propagated as soon as they are made by individual users. Such change requests (inserts and deletes) are sent as messages over a reliable network to all other users. Since the communication between the users is not instantaneous, the same messages may be received in different orders by different users. In this connection, elaborate algorithms have been devised that transform the change request messages received by a user in order to give the "intended meanings" of the requests and to ensure that all users have eventually (after all messages have been received) the same file content. dOPT [6] was the first such algorithm. It was later corrected in adOPTed [12] and GOT/GOTO [15,14]. Another such system can be found in [13]. (The partial update problem is also related to database transactions. We looked into database transaction literature and spoke to some experts in the field but we have not found articles addressing the partial update problem as we see it.)

Our study of partial updating of sequences here is by no means exhaustive but it includes those operations that we find most natural. In particular, we study a three-parameter substitution operation: given a file, replace the file segment of length ℓ that starts at position p with sequence s. We don't replace segments of the form "from a position p to the end of the sequence" but it would be easy to do so. We study also modifications where the segment to be replaced consists of just one position p and the replacement sequence contains just one element but the new element may depend on the old element at position p. We didn't investigate the more general case when a multi-element replacement sequence s depends on the content of a multi-element arena. This could be a good topic for future work (by us or somebody else).

In Section 4, we address partial updates of labeled ordered trees. We use the direct product of applicative algebras and the fixed-point operator to compose an applicative algebra over labeled ordered trees from the applicative algebra of labels and the applicative algebra of sequences. The applicative algebra over labeled ordered trees contains the natural operations of insertion, deletion and label alteration.

This paper is self-contained. One exception is the proof of Proposition 25 that uses the terminology of [10]. That proposition establishes that the applicative algebras over sets and maps, defined in [10], are distributive. The reader not interested in these two particular applicative algebras can skip the proof; nothing in the rest of the paper depends on it.

1.1 Acknowledgments

This paper was inspired by the work of the group on Foundations of Software Engineering in Microsoft Research. We thank Andreas Blass for useful discussions. Referee reports were helpful as well.

2 A Framework for Partial Updates

2.1 Preliminaries

We recall some useful definitions and establish some terminology and notation. The sign $\rightleftharpoons$ will mean equal by definition.

Multisets We use double curly braces for multisets. For example, $\{\!\{7, 7, 11\}\!\}$ is the multiset that contains 7 with multiplicity 2 and 11 with multiplicity 1 (and 13 with multiplicity 0). The cardinality of a multiset is the sum of the multiplicities. The underlying set of a multiset M is its *domain*. The domain of $\{\!\{7, 7, 11\}\!\}$ is $\{7, 11\}$.

Then *sum* $A+B$ of multisets A and B is the multiset C such that the domain $\mathtt{dom}\,(C) = \mathtt{dom}\,(A) \cup \mathtt{dom}\,(B)$ and $\mathtt{multiplicity}_C(x) = \mathtt{multiplicity}_A(x) + \mathtt{multiplicity}_B(x)$ for every $x \in \mathtt{dom}\,(C)$. If B is a submultiset of A (which means that every element of B occurs in A with the same or larger multiplicity) then $A - B$ is the multiset C such that $\mathtt{dom}\,(C) \subseteq \mathtt{dom}\,(A)$ and $\mathtt{multiplicity}_C(x) = \mathtt{multiplicity}_A(x) - \mathtt{multiplicity}_B(x)$.

It is convenient to view sets as special multisets where every element occurs with multiplicity 1. The symbol $\emptyset$ for the empty set will be used to denote the empty multiset as well. But notice that the sum operation differs from the union operation over sets. For example $\{1, 2\} + \{2, 3\} = \{\!\{1, 2, 2, 3\}\!\}$ while $\{1, 2\} \cup \{2, 3\} = \{1, 2, 3\}$.

Functional Composition The functional composition $g \circ f$ of unary functions f, g is defined as usual: $(g \circ f)(x) \rightleftharpoons g(f(x))$. In general $\mathtt{dom}\,(g \circ f) = \{x \in \mathtt{dom}\,(f) : f(x) \in \mathtt{dom}\,(g)\}$.

2.2 Applicative Algebras

Definition 1 An *applicative algebra* A has three constituents satisfying certain conditions.

Elements The first constituent is a datatype T with a distinguished element $\bot$ and at least one additional element. Elements of T are *elements* of A, and $\bot$ is the *trivial element*.

Particles The second constituent is a monoid of total unary operations over T with functional composition and the identity operation *id*. These operations are *particles* of A. And there is a *trivial particle* denoted $\bot$ as well. The following conditions are satisfied.

AA0 $f(\bot) = \bot$ for every particle f, and $\bot(x) = \bot$ for every element x.

Parallel Composition The third constituent is an operation $\mathbf{\Omega}$ that, given an arbitrary finite multiset of particles, produces a particle. $\mathbf{\Omega}M$ is the *parallel composition* of the multiset M of particles. The following conditions are satisfied.

AA1 $\mathbf{\Omega}\{\!\{f\}\!\} = f$, and $\mathbf{\Omega}(M + \{\!\{id\}\!\}) = \mathbf{\Omega}M$, and $\mathbf{\Omega}(M + \{\!\{\bot\}\!\}) = \bot$. $\square$

Corollary 2
1. $\mathbf{\Omega}\,\emptyset = id$.
2. $f \circ \bot = \bot \circ f = \bot$.

PROOF.
1. $\mathbf{\Omega}\,\emptyset = \mathbf{\Omega}(\emptyset + \{\!\{id\}\!\}) = \mathbf{\Omega}\{\!\{id\}\!\} = id$.
2. Check that the equalities hold at every point. □

Remark 3 (Algebra) In more algebraic terms, A is a multi-sorted algebra with at least three sorts: elements, particles and multisets of particles. T is an algebra in its own right, with nullary operation $\bot$ and possibly some additional operations. The additional operations may involve some auxiliary sorts. For example, an algebra of sequences may have an additional operation `length` from sequences to integers. The particles operate over the main sort elements only.

Remark 4 (Notation) The particles together with the functional composition form a monoid. In some cases, it is convenient to think of the parallel composition as addition and to use the additive notation $\sum M$ instead of $\mathbf{\Omega}M$. For example, when elements are real numbers and particles have the form $x \mapsto x + r$. In some other cases, it is convenient to use the multiplicative notation $\prod M$. Neither additive nor multiplicative notation is always appropriate, and so we use a neutral notation.

Example 5 The particle type consists of the identity particle and the family of `overwrite`$[y]$ particles

$$\texttt{overwrite}[y] : x \mapsto \begin{cases} y & \text{if } x \neq \bot \\ \bot & \text{otherwise} \end{cases}$$

Note that `overwrite`$[\bot]$ is the trivial particle. Define $\mathbf{\Omega}M$ as follows.

- If M contains neither $\bot$ nor overwrite particles then $\mathbf{\Omega}M = id$.
- If M contains $\bot$ or distinct overwrite particles then $\mathbf{\Omega}M = \bot$.
- If M contains an overwrite particle f and neither $\bot$ nor other overwrite particles then $\mathbf{\Omega}M = f$. □

It is typical that particles come in families. A particular particle f of a family F is given by a tuple of parameters which will be called the *control* of f in F. The control of f will be shown in brackets and the argument of f in parenthesis e.g. `overwrite`$[y](x)$.

Definition 6 A multiset M of particles is *consistent* if $\mathbf{\Omega}M \neq \bot$.

The trivial element $\bot$ used above is just a convenient device to make partial operations total. There is a natural version of the notion of applicative algebras without $\bot$.

Definition 7 A *punctured applicative algebra* A has three constituents.

- A nonempty datatype T of *elements*.
- A set of partial unary operations (*particles*) over T together with functional composition and the identity operation *id*.

- A partial operation (*parallel composition*) $\mathbf{\Omega}$ that, given a finite multiset of particles in its domain, produces a particle. The following conditions are satisfied.
  PAA1 $\mathbf{\Omega}\{\!\{f\}\!\} = f$, and $\mathbf{\Omega}(M + \{\!\{id\}\!\}) = \mathbf{\Omega}M$. $\square$

It is easy to see that removing the trivial element from an applicative algebra A turns it into a punctured applicative algebra A_0, a punctured version of A. The converse is also true. Extending a punctured applicative algebra A_0 with a fresh element, intended to play the role of the trivial element $\perp$, turns A_0 into an applicative algebra A, the *filled version* of A_0. If A_0 has the nowhere defined particle, it will play the role of the trivial particle; otherwise the trivial particle should be added. The new, total parallel composition operation $\mathbf{\Omega}$ extends the old, partial parallel composition operation $\mathbf{\Omega}_0$ by producing the trivial particle if the given multiset M contains the trivial particle or if M does not contain the trivial particle but $\mathbf{\Omega}_0 M$ is undefined. Since A_0 and A are so closely related, we will ignore the difference between them.

2.3 A Context for Partial Updates

Partial updates are used extensively in AsmL. We describe that context, abstracting of AsmL-specific details in line with our intention to make this paper self-contained.

Transactions The given computer system $\mathbf{S}$ evolves by means of transactions. A transaction may involve interaction with the outside world. A run of the system can be defined as a sequence of transactions. A transaction may fail. If it succeeds, it takes system $\mathbf{S}$ from the current state to the next state. In the case of AsmL, if a transaction fails, an exception is thrown which may or may not be caught. Here we do not address exception handling.

Locations We assume that the given state consists of disjoint locations with values stored in them. Each location ℓ has a particular type T. Only values of type T or its subtypes can be stored at ℓ. (We will not be dealing with subtyping in the sequel.) It is technically convenient to assume that transactions do not create new locations. The assumption may seem restrictive but it isn't: just pretend that the locations created by a transaction had existed but were inactive.

Partial Updates A *partial update* $\mathrm{PU}(\ell, f)$ is given by a location ℓ of some type T and a particle f over T. We say that $\mathrm{PU}(\ell, f)$ is a *partial update of* ℓ and that it *modifies* ℓ. If U is a multiset of partial updates then $\mathrm{Loc}(U)$ is the set of locations modified by partial updates in U.

Integration of Partial Updates Let U be a multiset of partial updates. For each $\ell \in \mathrm{Loc}(U)$, let $M_\ell(U)$ be the multiset $\{\!\{f : \mathrm{PU}(\ell, f) \in U\}\!\}$ of particles in U modifying

ℓ. If any $\mathbf{\Omega}M_\ell(U) = \bot$ then the integration of U fails. Otherwise the integration of U results in the assignment $\mathbf{\Omega}U$ of particles to locations:

$$(\mathbf{\Omega}U)(\ell) \rightleftharpoons \mathbf{\Omega}M_\ell(U) \quad \text{where} \quad \ell \in \mathrm{Loc}(U)$$

The New State As a given transaction executes, it generates partial updates. The execution may fail. Suppose that the execution succeeds. Let U be the multiset of all partial updates generated by the transaction, let ℓ range over $\mathrm{Loc}(U)$, and let x_ℓ be the content of ℓ at the current state. If U doesn't integrate, the transaction fails. Suppose that U is integrable. Let f_ℓ be the particle $(\mathbf{\Omega}U)(\ell)$, and let $y_\ell = f_\ell(x_\ell)$. If any $y_\ell = \bot$ then the transaction fails. Otherwise the transaction succeeds and a new state is constructed by replacing x_ℓ with y_ℓ for all $\ell \in \mathrm{Loc}(U)$. The content of any location outside $\mathrm{Loc}(U)$ does not change.

According to the definition of applicative algebras, if a transaction fails then all parent transactions fail as well. (More sophisticated failure treatments can be conceived. As we mentioned above, AsmL incorporates exception handling. But failure treatment is beyond the scope of this paper.)

Nested Transactions A global transaction may include auxiliary transactions. Each of them may be again a composition of transactions, and so on. Let us call the top transaction of that hierarchy *global*; all other transactions in the hierarchy are *constituents* of the global transaction.

Each successful transaction in the hierarchy computes a multiset of partial updates and then integrates them into an assignment. There is, however, an important distinction between the global transaction and its constituents. The global transaction executes its assignment producing a new state of the computer system $\mathbf{S}$. A constituent transaction reports its assignment to the parent transaction. If any constituent transaction fails then the parent transaction fails as well. It follows that the global transaction fails if any of the constituent transactions does.

A constituent transaction does not necessarily operate over the current state of the system $\mathbf{S}$. It may operate over a virtual state of $\mathbf{S}$. What these virtual states are will be clarified immediately.

Composing Transactions We restrict attention to the following two ways to combine transactions: parallel composition and sequential composition. (In AsmL there are numerous ways to program either composition.)

First we consider the case when the given transaction τ (not necessarily a global transaction) is the parallel composition of transactions $\tau_1, \ldots, \tau_k$ where $k \geq 2$. If τ operates over a (possibly virtual) state X then τ_i operates over some extension X_i of X. Think about the recently activated locations of X_i as the scratch paper of τ_i. If the scratch paper is not empty then X_i is a virtual state even if X was the actual state, that is the current state of $\mathbf{S}$.

If τ_i succeeds, it computes a multiset U_i of partial updates that modify X (rather than its own scratch paper), then it integrates U_i into the assignment $\mathbf{\Omega}U_i$, and then it reports $\mathbf{\Omega}U_i$ to τ. If all transactions $\tau_1, \ldots, \tau_k$ succeed then the transaction τ forms the multiset U of all partial updates (ℓ, f) such that $f = (\mathbf{\Omega}U_i)(\ell)$ for some i. Then it integrates U and then either reports or executes the assignment $\mathbf{\Omega}U$.

Second we consider the case when the given transaction τ is the sequential composition of transactions τ_1 and τ_2. It suffices to consider the composition of two constituent transactions even though the language for programming transactions may use e.g. the while command to produce sequences of constituent transactions. The reason is that every successful global transaction contains only finitely many constituent transactions; there is no need for any limit operation.

Let X be a (possibly virtual) state over which τ computes. Transaction τ_1 computes over an extension of X. If it is successful then it produces a set U_1 of partial updates that modify X. τ_1 turns X into an intermediate virtual state X' with the same locations as X. Transaction τ_2 computes over the virtual state X'. If it is successful then it produces a set U_2 of partial updates of locations in X'. Then τ computes the multiset (in fact the set) U of partial updates (ℓ, f) such that one of the following conditions holds.

- $\ell \in \mathrm{Loc}(U_1) - \mathrm{Loc}(U_2)$ and $f = (\mathbf{\Omega}U_1)(\ell)$.
- $\ell \in \mathrm{Loc}(U_2) - \mathrm{Loc}(U_1)$ and $f = (\mathbf{\Omega}U_2)(\ell)$.
- $\ell \in \mathrm{Loc}(U_1) \cap \mathrm{Loc}(U_2)$ and $f = (\mathbf{\Omega}U_2)(\ell) \circ (\mathbf{\Omega}U_1)(\ell)$.

Then it integrates U and then either reports or executes the assignment $\mathbf{\Omega}U$.

Finally, any transaction in the transaction hierarchy can interact with the outside world (which opens an additional venue for inter-transaction exchange of information). For example, it can send and receive messages or print a file on paper.

2.4 Properties of Applicative Algebras

Let A be an applicative algebra. Define $f \times g = \mathbf{\Omega}\{\!\{f, g\}\!\}$. The choice of multiplicative notation is arbitrary but convenient for exposition.

Definition 8 A is *associative* if the multiplication operation $f \times g$ is associative.

Definition 9 A is *distributive* if $\mathbf{\Omega}$ distributes over the multiset sum operation:

$$\mathbf{\Omega}(M_1 + \cdots + M_n) = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \ldots, \mathbf{\Omega}M_n\}\!\}$$

for all multisets $M_1, \ldots, M_n$ of particles. A is *conditionally distributive* if $\mathbf{\Omega}(M_1 + \cdots + M_n) = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \ldots, \mathbf{\Omega}M_n\}\!\}$ provided that $\mathbf{\Omega}(M_1 + \cdots + M_n) \neq \bot$.

A referee noted that this distributivity notion is referred to as generalized associativity e.g. in [4, Section V.5].

If A is associative then the binary operation $f \times g$ gives rise to the multiset operation $\prod\{\!\{f_1, \ldots, f_n\}\!\} = f_1 \times \cdots \times f_n$ on particles where $\prod\{\!\{f\}\!\} = f$ and $\prod \emptyset = id$. However, this product $\prod$ does not necessarily coincide with the original multiset operation $\mathbf{\Omega}$.

Lemma 10 *A is distributive if and only if it is associative and $\prod$ coincides with $\mathbf{\Omega}$.*

PROOF. The if direction holds because, in any commutative monoid, the product operation distributes over the multiset sum operation. For example,

$$\begin{aligned}&\textstyle\prod(\prod\{\!\{f_1, f_2\}\!\}, \prod\{\!\{f_3, f_4\}\!\}) = (f_1 \times f_2) \times (f_3 \times f_4)\\ &= f_1 \times f_2 \times f_3 \times f_4 = \textstyle\prod(\{\!\{f_1, f_2\}\!\} + \{\!\{f_3, f_4\}\!\})\end{aligned}$$

Suppose that A is distributive. Then

$$\begin{aligned}&(f \times g) \times h = \mathbf{\Omega}\{\!\{\mathbf{\Omega}(f, g), \mathbf{\Omega}\{\!\{h\}\!\}\}\!\} = \mathbf{\Omega}(\{\!\{f, g\}\!\} + \{\!\{h\}\!\})\\ &= \mathbf{\Omega}(\{\!\{f\}\!\} + \{\!\{g, h\}\!\}) = \mathbf{\Omega}(\mathbf{\Omega}\{\!\{f\}\!\}, \mathbf{\Omega}\{\!\{g, h\}\!\}) = f \times (g \times h)\end{aligned}$$

Now check the equality $\mathbf{\Omega} f_1, \ldots, f_n = \prod f_1, \ldots, f_n$ by induction on n. $\square$

Example 11 Distributivity does not follow from associativity alone. Indeed, consider Example 5 and modify the third clause in the definition of $\mathbf{\Omega}$ as follows:

- If M contains one or two copies of an overwrite particle f and neither $\perp$ nor other overwrite particles then $\mathbf{\Omega}M = f$, but if M contains at least three copies of an overwrite particle then $\mathbf{\Omega}M = \perp$.

It is easy to check that the resulting applicative algebra is associative but not distributive. If f is any overwrite particle, we have

$$\mathbf{\Omega}(\{\!\{f, f\}\!\} + \{\!\{f\}\!\}) = \mathbf{\Omega}\{\!\{f, f, f\}\!\} = \perp \neq f = \mathbf{\Omega}(\mathbf{\Omega}\{\!\{f, f\}\!\}, \mathbf{\Omega}\{\!\{f\}\!\}). \quad \square$$

Moreover, an applicative algebra can be conditionally distributive but not associative. See Propositions 48 and 49 in this connection.

Definition 12 An applicative algebra A is *pervasive up* if the following condition holds for all nontrivial particles $f_1, \ldots, f_n$:

$$\text{if } f_i \times f_j \neq \perp \text{ for all } i < j \text{ then } \mathbf{\Omega}\{\!\{f_1, \ldots, f_n\}\!\} \neq \perp.$$

A is *pervasive down* if the following condition holds for any multisets M_1, M_2 of particles:

$$\text{if } M_1 \text{ is a submultiset of } M_2 \text{ and } \mathbf{\Omega}M_2 \neq \perp \text{ then } \mathbf{\Omega}M_1 \neq \perp. \quad \square$$

Corollary 13 *Suppose that A is pervasive up and down. Then, for every multiset M of nontrivial particles, the following are equivalent:*

(1) M is consistent.
(2) M is pairwise consistent.

Lemma 14 *Suppose that A is conditionally distributive. Then it is pervasive down. And if a multiset $\{\!\{f, g, h\}\!\}$ is consistent then $(f \times g) \times h = f \times (g \times h)$.*

PROOF. Let M_2 be a consistent multiset of particles, and let M_1 be a submultiset of M_2. Then $\mathbf{\Omega}M_2 = (\mathbf{\Omega}M_1) \times (\mathbf{\Omega}(M_2 - M_1)) \neq \bot$. It follows that $\mathbf{\Omega}M_1 \neq \bot$.

Further, $(f \times g) \times h = \mathbf{\Omega}\{\!\{\mathbf{\Omega}\{\!\{f, g\}\!\}, \mathbf{\Omega}\{\!\{h\}\!\}\}\!\} = \mathbf{\Omega}\{\!\{f, g, h\}\!\} = \mathbf{\Omega}\{\!\{\mathbf{\Omega}\{\!\{f\}\!\}, \mathbf{\Omega}\{\!\{g, h\}\!\}\}\!\} = f \times (g \times h)$. $\square$

Example 15 We construct a distributive applicative algebra that is not pervasive up. Elements are $\{a, b, c, \bot\}$. Particles are operations f_S where S is any subset of nontrivial elements; $f_S(x) = x$ if x in S and $f_S(x) = \bot$ otherwise. The trivial particle $\bot = f_\emptyset$ and $id = f_S$ where $S = \{a, b, c\}$. For every multiset $M = \{\!\{f_{S_1}, \ldots, f_{S_n}\}\!\}$ of particles, define $\mathbf{\Omega}M = f_U$ where $U = S_1 \cap \cdots \cap S_n$. It is easy to see that this applicative algebra is distributive. Let $U = \{a, b\}$, $V = \{b, c\}$, $W = \{c, a\}$, then f_U, f_V, f_W are pairwise consistent but $\mathbf{\Omega}\{\!\{f_U, f_V, f_W\}\!\} = \bot$. $\square$

It follows that if $M = \{\!\{f_1, \ldots, f_n\}\!\}$ is a consistent multiset of particles of a conditionally distributive applicative algebra then the product $\prod M = f_1 \times \cdots \times f_n$ is well defined.

Lemma 16 *Suppose that A is conditionally distributive. And let $M = \{\!\{f_1, \ldots, f_n\}\!\}$ be any consistent multiset of particles. Then $\mathbf{\Omega}M = \prod M$.*

PROOF. Induction on n. $\square$

Definition 17 A consistent multiset M of particles is *(pointwise) coherent* if $(\mathbf{\Omega}M)(x) \neq \bot$ for every x such that $f(x) \neq \bot$ for all $f \in M$.

Proposition 18 *Suppose that A is conditionally distributive. If every consistent two-particle multiset is coherent then every consistent multiset $M = \{\!\{f_1, \ldots, f_n\}\!\}$ is coherent.*

PROOF. Induction on n. The case $n \leq 1$ is trivial. Suppose that $n > 1$ and the proposition has been proved for $m = n - 1$. Suppose that $f_1(x), \ldots, f_n(x)$ are nontrivial. Let $M_1 = \{\!\{f_1, \ldots, f_m\}\!\}$, $M_2 = \{\!\{f_n\}\!\}$, $f = \mathbf{\Omega}M_1$, $g = \mathbf{\Omega}M_2$ and $h = \mathbf{\Omega}M$. We need to prove that $h(x) \neq \bot$.

By Lemma 14, M_1 is consistent. By the induction hypothesis, $f(x) \neq \bot$. By the conditional distributivity, $h = \mathbf{\Omega}(M_1 + M_2) = \mathbf{\Omega}(\mathbf{\Omega}(M_1), \mathbf{\Omega}(M_2)) = \mathbf{\Omega}(f, g) = f \times g$. Now use the pairwise coherence. $\square$

2.5 Functional Applicative Algebras

Sometimes the parallel composition $\mathbf{\Omega}$ is defined in terms of functional composition.

Definition 19 A *functional applicative algebra* is an applicative algebra where

$$\mathbf{\Omega}\{\!\{f_1, \ldots, f_n\}\!\} = f_1 \circ f_2 \circ \cdots \circ f_n$$

if all f_i, f_j commute, and $\mathbf{\Omega}\{\!\{f_1, \ldots, f_n\}\!\} = \bot$ otherwise.

Note that

$$f \times g = \begin{cases} g \circ f & \text{if} \quad g \circ f = f \circ g, \\ \bot & \text{if} \quad g \circ f \neq f \circ g. \end{cases}$$

Proposition 20 *Every functional applicative algebra is conditionally distributive.*

PROOF. Suppose that $\mathbf{\Omega}M \neq \bot$ and let $M = M_1 + \cdots + M_n$. We need to prove that $\mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \ldots, \mathbf{\Omega}M_n\}\!\} = \mathbf{\Omega}M$. We illustrate the proof on the case $n = 2$. Let $M_1 = \{\!\{f_1, \ldots, f_j\}\!\}$ and $M_2 = \{\!\{g_1, \ldots, g_k\}\!\}$. Since $\mathbf{\Omega}M$ is consistent, the functional composition of the members of M in any order produces $\mathbf{\Omega}M$. Therefore $\mathbf{\Omega}M_1 \circ \mathbf{\Omega}M_2 = \mathbf{\Omega}M_2 \circ \mathbf{\Omega}M_1 = \mathbf{\Omega}M$, so that $\mathbf{\Omega}M_1 \times \mathbf{\Omega}M_2 = \mathbf{\Omega}M$ and $\mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \mathbf{\Omega}M_2\}\!\} = \mathbf{\Omega}M$. □

However, a functional applicative algebra is not necessarily distributive. Furthermore, it is not necessarily associative, even though functional composition is associative. In fact, the functional applicative algebra over counters from [10] is not associative. A self-explanatory counterexample is given by the following three particles: f is an overwrite, g is an increment by one and h is a decrement by one. We have $f \circ g = f \neq g \circ f$ and so $\bot = f \times g = (f \times g) \times h$. On the other hand, $g \circ h = h \circ g = id$ and so $f \times (g \times h) = f \times id = f \neq \bot$.

2.6 Apt Functional Applicative Algebras

A priori, functional consistency is hard to verify, but there is an important class of functional applicative algebras where the verification is easy. The following definition is borrowed from [10] except that we didn't use the term "applicative algebra" there. Recall that particles f, g commute if $g \circ f = f \circ g$. Say that particles f, g *malcommute* if $(g \circ f)(x) \neq (f \circ g)(x)$ for all nontrivial elements x.

Definition 21 A functional applicative algebra A is *apt* if

- every nontrivial particle maps nontrivial elements to nontrivial elements, and
- every nontrivial particles f, g either commute or malcommute. □

In order to find out whether nontrivial particles f, g of an apt functional applicative algebra commute or malcommute, it suffices to check the equality $(g \circ f)(x) = (f \circ g)(x)$ for any nontrivial element x.

Lemma 22 ([10, Lemma 7.3]) *Let A be an apt functional applicative algebra and let M be a multiset of particles of A. Then M is functionally consistent if and only if every two members of M commute.*

Proposition 23 *Every apt functional applicative algebra A has the following properties.*

(1) Every consistent multiset is coherent.

(2) A is pervasive up and down.

PROOF. Part 1 is straightforward. To prove part 2, use Lemma 22. □

Lemma 24 *For every apt functional applicative algebra A, the following properties are equivalent:*

(1) A is distributive.

(2) For every nontrivial particles f, g, h, if f, g malcommute and g, h commute, then f and $(g \circ h)$ malcommute.

PROOF.

1 $\Rightarrow$ *2.* Assume *1.* Then A is associative. Suppose that f, g malcommute and g, h commute. Then

$$f \times (g \circ h) = f \times (g \times h) = (f \times g) \times h = \bot \times h = \bot$$

It follows that f and $g \circ h$ malcommute.

2 $\Rightarrow$ *1.* Assume *2.* First we prove that A is associative. Given T particles f, g, h, we check that $(f \times g) \times h = f \times (g \times h)$. This is obvious if one of the particles is $\bot$. So suppose that all the particles f, g, h are nontrivial particles. Several cases arise.

- f, g commute and g, h commute and f, h commute. So every two members of $M \rightleftharpoons \{\!\{f, g, h\}\!\}$ commute. Then we know by Lemma 22 that M is functionally consistent and thus $(f \circ g) \circ h = h \circ (f \circ g) \neq \bot$ and $(g \circ h) \circ f = f \circ (g \circ h) \neq \bot$. This means that $f \circ g = f \times g$ commutes with h and $g \circ h = g \times h$ commutes with f. Thus $(f \times g) \times h = (f \circ g) \circ h = f \circ (g \circ h) = f \times (g \times h)$.
- f, g commute and g, h commute and f, h malcommute. For every nontrivial element y of type T, $(f \circ h)(y) \neq (h \circ f)(y)$. In particular, for every nontrivial x, $(f \circ h)(gx) \neq (h \circ f)(gx)$. Hence $(f \circ (h \circ g))(x) = (f \circ h)(gx) \neq (h \circ f)(gx) = (h \circ f \circ g)(x) = (h \circ g \circ f)(x) = ((h \circ g) \circ f)(x)$ for all nontrivial x of type T, so that f and $g \circ h = h \circ g$ malcommute. By symmetry, $f \circ g$ and h malcommute. Thus $f \times (g \times h) = \bot = (f \times g) \times h$.
- f, g commute and g, h malcommute. Then $f \times (g \times h) = f \times \bot = \bot$. By *2*, $(f \times g) \times h = \bot$.
- f, g malcommute and g, h commute. Then $(f \times g) \times h = \bot \times h = \bot$. By *2*, $f \times (g \times h) = \bot$.
- f, g malcommute and g, h malcommute. Then $(f \times g) \times h = \bot \times h = \bot$ and $f \times (g \times h) = f \times \bot = \bot$.

Thus the parallel composition is associative. By Lemma 10, it suffices to prove that $\mathbf{\Omega}\{\!\{f_1, \ldots, f_n\}\!\} = f_1 \times \cdots \times f_n$ for any particles $f_1, \ldots, f_n$. Without loss of generality, all n particles are nontrivial. If the n particles pairwise commute then the equality follows from the definition of parallel composition. So assume that not all n particles pairwise commute.

It suffices to prove the following: $g_1 \times \cdots \times g_m = \bot$ if g_1 and g_m malcommute. We prove this by induction on m. The case $m \leq 2$ is obvious; use the definition of parallel composition. So assume that $m > 2$. If g_{m-1} malcommutes with g_m then $g_{m-1} \times g_m = \bot$ and therefore $g_1 \times \cdots \times g_m = \bot$. So assume that g_{m-1} commutes with g_m. Then swap g_{m-1} and g_m and use the induction hypothesis. $\square$

Proposition 25
1. The applicative algebra over maps, defined in [10], is distributive.
2. The applicative algebra over sets, defined in [10], is distributive.

The proposition is of independent interest: the two algebras are most natural and are used in the implementation of AsmL. But it will not be used in rest of the paper. To save space, the proof uses the terminology of [10], though the proof of the second part of the proposition can be understood without consulting [10].

PROOF.

1. It suffices to check the condition *2* of Lemma 24. Let f, g, h be ranked transformers such that f, g malcommute and g, h commute. We prove that f and $g \circ h$ malcommute.

We say that a transformer f is *map-valued*, if $f(m)$ is a map for every point m. Note that if f is not map-valued then it is of rank 0 and so $f(x) = a$ for all points x and for some non-map point a. Recall that a map-valued transformer f has a controller c with associated transformers c_x such that $c_x(mx) = (fm)(x)$ for all points m and x. Theorem 11.12 in [10] implies that two map-valued transformers f, g with controllers c, d respectively commute if and only if c_x and d_x commute for every point x.

The proof goes by induction on rank(g). Suppose g has rank zero. Recall that a zero-ranked transformer is a constant function, so $g \circ h = g$ as g, h commute. As f, g malcommute, so do f and $g = g \circ h$. So let g be of rank > 0. Then g is map-valued. As $g \circ h = h \circ g$, h returns a map at least at some points. It follows that h is map-valued. Indeed, if $h(x) = a$ and a is not a map for some x, then rank$(h) = 0$ and so $h(x) = a$ for all points x which is impossible.

If f is not map-valued, then f and $g \circ h$ malcommute because $f \circ (g \circ h)$ is not a map while $(g \circ h) \circ f$ is a map. So assume f is map-valued. Let c, d, e be the controllers of f, g, h respectively. As f, g malcommute, there is a particular x such that c_x, d_x malcommute. Similarly, all d_y, e_y commute because g, h commute; in particular d_x and e_x commute. We know that rank$(d_x) <$ rank(g). By induction hypothesis, c_x and $d_x \circ e_x$ malcommute. Hence f and $g \circ h$ malcommute.

2. It suffices to check the condition *2* of Lemma 24. Let f, g, h be set particles. Suppose g is an overwrite particle. Then $g \circ h = g$. As f, g malcommute, so do f and $g = g \circ h$. So let g be an insert-and-remove operation, and so every $((g \circ h) \circ f)(x)$ is a set. So assume every $(f \circ (g \circ h))(x)$ is a set. f could be an insert-and-remove particle or an overwrite particle that produces some set a; in the second case, we can see f as a generalized insert-and-remove particle: given a set s, it removes all elements in s that do not belong to a and then inserts all elements of a. As f, g malcommute, there are two cases:

Case 1: There is an element x that f inserts and g removes. Since g, h commute, h cannot insert that x. Hence $x \in (f \circ (g \circ h))(x)$ and $x \notin ((g \circ h) \circ f)(x)$. Therefore f and $g \circ h$ malcommute.

Case 2: There is an element x that f removes and g inserts. Because g, h commute, h cannot remove that x. Hence $x \notin (f \circ (g \circ h))(x)$ and $x \in ((g \circ h) \circ f)(x)$. Therefore f and $g \circ h$ malcommute. $\square$

2.7 The Product of Applicative Algebras

Definition 26 (Product) We define the *product* $A \times B$ of applicative algebras A and B. Elements of $A \times B$ are pairs (x, y) where x, y are elements of A, B respectively. If x or y is trivial then (x, y) is identified with the trivial element in $A \times B$.

The particles of $A \times B$ are `alter` particles working componentwise.

$$\texttt{alter}[f,\, g\,] : (x,\, y) \mapsto (f(x),\, g(y))$$

where f, g are particles of A, B respectively. If f or g is trivial then $\texttt{alter}[f, g]$ is identified with the trivial particle in $A \times B$. The parallel composition is defined componentwise as well:

$$\mathbf{\Omega}\{\!\{\texttt{alter}[f_1, g_1], \ldots, \texttt{alter}[f_n, g_n]\}\!\} \rightleftharpoons \texttt{alter}[\mathbf{\Omega}\{\!\{f_1, \ldots, f_n\}\!\}, \mathbf{\Omega}\{\!\{g_1, \ldots, g_n\}\!\}] \quad \square$$

A referee noted that the construction of taking a product and then identifying all pairs that have either component equal to a certain specified element is a common one in topology and is called there the smash product.

Lemma 27

- *A particle* $\texttt{alter}[f, g]$ *is the identity particle if and only if both* f *and* g *are identity particles.*
- *The requirements AA0 and AA1 of the definition of applicative algebras are satisfied.*

PROOF. Straightforward. $\square$

Remark 28 One can define the sum (or disjoint union) $A + B$ of applicative algebras A and B that have only the trivial element in common. Elements of $A + B$ are elements of A or B. The new particles are A particles extended to produce $\perp$ at any element of B, and B particles extended to produce $\perp$ at any element of A, and the identity particle. We leave the details as an exercise. The sum operation will not be used in the sequel.

2.8 The Complexity of Parallel Composition

Suppose that $M = \{\!\{f_1, \ldots, f_n\}\!\}$ is a multiset of particles of an applicative algebra A. How hard is it to compute $\mathbf{\Omega}M$? Note that computing $\mathbf{\Omega}M$ allows us to decide whether M is consistent; just check whether $\mathbf{\Omega}M = \perp$. If M contains $\perp$ then $\mathbf{\Omega}M = \perp$. So let us assume that M does not contain $\perp$. Since the definition of applicative algebras is so general, there isn't much to say about the problem in full generality. But there are important special cases where the problem of computing arbitrary $\mathbf{\Omega}M$ reduces to the binary fragment, that is to the problem of computing $\mathbf{\Omega}M$ for multiset a M of cardinality two. In all particular cases that we studied, there are efficient algorithms for computing $\mathbf{\Omega}M$.

Functional Applicative Algebras Suppose that A is a functional applicative algebra, and let $M = \{\!\{f_1, \ldots, f_n\}\!\}$. Then $\mathbf{\Omega}M \neq \bot$ if and only if the compositions of $f_1, \ldots, f_n$ in all $n!$ possible orderings give the same nontrivial result.

Suppose, however, that A is apt. By Lemma 22, $\mathbf{\Omega}M \neq \bot$ if and only if M does not contain $\bot$ and every two members of M commute. This requires $n(n-1)/2$ commute checks. And if $\mathbf{\Omega}M \neq \bot$ then $\mathbf{\Omega}M = f_1 \circ \cdots \circ f_n$ which requires $n-1$ functional compositions.

Pervasive Applicative Algebras Suppose that A is an applicative algebra that is pervasive up and down. Assume that we know how to check whether $\mathbf{\Omega}M = \bot$ for every M of cardinality two. This does not allow us to compute an arbitrary $\mathbf{\Omega}M$ but it is enough to check the consistency of an arbitrary M. Namely, M is consistent if and only if it does not contain $\bot$ and $\mathbf{\Omega}M' \neq \bot$ for all submultisets M' of cardinality two. This requires $n(n-1)/2$ binary consistency checks where n is the cardinality of M.

(Conditionally) Distributive Applicative Algebras First suppose that A is distributive, and let $M = \{\!\{f_1, \ldots, f_n\}\!\}$. By Lemma 10, A is associative and $\mathbf{\Omega}M = f_1 \times \cdots \times f_n$. In this case, only $n-1$ binary parallel compositions suffice to compute $\mathbf{\Omega}M$. By Proposition 25, the applicative algebras over sets and maps are distributive.

Second suppose that A is conditionally distributive and pervasive up. By Lemma 14, A is pervasive down as well and therefore, as above, we have an algorithm for checking the consistency of a particle multiset. But Lemma 16 gives us an algorithm for computing parallel composition.

The Applicative Algebra over Counters The applicative algebra over counters, constructed in [10], is not distributive or even associative, but there is an efficient algorithm for checking consistency. The algorithm can be understood without reading [10]. Let $M = \{\!\{f_1, \ldots, f_n\}\!\}$ be a multiset of particles. The only particles are increment and overwrite particles. An increment particle has an integer parameter (the control) p that is added to the counter. Overwrite particles were described in Example 5. If M has at least one non-zero increment and an overwrite particle then $\mathbf{\Omega}M = \bot$. If M has two distinct overwrites, then $\mathbf{\Omega}M = \bot$. Otherwise $\mathbf{\Omega}M = f_1 \circ \cdots \circ f_n$.

The Applicative Algebra over Sequences The applicative algebra over sequences, introduced in the next section, is not associative either, but again there is an efficient algorithm for parallel composition; see Subsection 3.4.

3 Case Study: Sequences

Sequences play a most important role in computing. Text processors work with sequences of characters (files). A database table is a sequence of records. The reader will easily come up with additional examples.

The problem of merging various modifications of the same sequence is important as well. Here is one example. Text processors perform basic operations on the given file, like inserting a string, deleting a part of the file, altering a part of a file. Two or more people may independently change copies of the same file. Their changes may involve numerous basic operations. And the changes need to be integrated. In this connection we looked at

- The UNIX diff and patch commands documented in UNIX manuals. See [8] for example.
- The merge-and-compare feature of Microsoft Word [11].

The diff command takes two files as its input: an original file and a derived file. Here a file is a sequence of lines. The command computes the difference of the derived document from the original one in terms of inserted, deleted and altered segments of the file, each segment consisting of some lines. The result is stored as a so-called patchfile. The patch command takes such a patchfile and another file as its input. It is expected that the other file is similar to the original file, so that there is a so-called horizon of at least two unmodified lines around every insertion, deletion and alteration described in the patchfile. Because of this tolerance, one can try to apply several patchfiles, one after another, to a file. If the two-line horizon rule is violated, then patches may be rejected. A rejection may occur even if the modifications are consistent but too close to each other.

The "Merge and Compare" feature of Microsoft Word is more sophisticated and complicated. The precise algorithms are not publicly available, but our experiments suggest the following. When a document is modified, not only the final result is saved but also the relative changes are saved. Relative changes can be insertions and deletions as well as alterations of the styles (font styles, paragraph styles, etc.) of file segments. Starting from the original document, one can merge several sets of relative changes. Deletions and insertions are highlighted. Microsoft Word relies on the detailed description of the relative changes and does not require horizons. Even contradicting modifications are merged; the modifications are applied sequentially.

In this section, we propose a new framework for partial updates of sequences. It is clean and precise. Consistent partial updates are always integrated, and inconsistent partial updates are always detected. We provide efficient algorithms that handle parallel and sequential compositions of partial updates and detect contradictions. Admittedly, our sequence-modification operations can be made richer; this is a topic for future work.

We study partial updates of sequences, that is finite sequences. It could be sequences of characters but we take a more general point of view. Fix an arbitrary applicative algebra A that contains at least two nontrivial elements. We will construct an applicative algebra $SEQ(A)$. The elements of $SEQ(A)$ are finite sequences of nontrivial elements of A as well as the trivial sequence $\perp$. Note that the empty sequence differs from $\perp$ and thus is a nontrivial sequence. The particles and the parallel composition of particles are defined below.

Notation If s is a sequence (any sequence, not necessarily an element of $SEQ(A)$) then $\#s$ is the length of s and s_i denotes the ith element of s where $i \in \{0, \ldots, \#s-1\}$. If s is a sequence of elements of some datatype D and p is a natural number such that $\#s > p$ and a is an element of datatype D, then $s(p/a)$ is the sequence obtained from s by setting the element in position p to a. We denote the concatenation of two sequences x and y as $x \cdot y$ or simply xy. We write ϵ for the empty sequence.

3.1 Natural Operations on Sequences

In our opinion, the most natural operations on sequences are insertion, deletion and alteration. The sequence being modified will be often called a *file*.

Positions A file of length n has n distinct *positions*. We count positions from left to right starting from number 0. For example, the file *abb* has three positions. It has a in position 0 and b in positions 1 and 2.

Places The concept of *place* arises because of insertions. A sequence can be inserted into a file x of length n in the following places. If $n = 0$, then there is only one place. If $n > 0$, then it can be inserted at the beginning of x (before position 0), between any two adjacent positions of x, and at the end of x (after position $n-1$). In any case, a file x of length n has $n+1$ distinct *places*. We count places from left to right starting from number 0. For example, the file *abb* has four places. Place 0 is before a, that is before position 0. Place 1 is between a and the first b, that is between positions 0 and 1. Place 2 is between the two b's, that is between positions 1 and 2. And place 3 is after the second b, that is after position 2.

Using the notions of position and place, we can describe the three natural modifications of a given file.

- Insert a given sequence s into a given place of the file.
- Delete the part of the file that is given by a segment of positions.
- Alter the element of applicative algebra A at a given position by a given A particle.

It is convenient to view the operations of insertion and deletion as special cases of a more general operation of substitution. Then there are only two basic families of file modifications: substitutions and alterations.

3.2 Prime Particles

We define sequence particles in two steps. In this subsection, we define three families of *prime particles*: substitutions, alterations, and position particles. (A position particle checks that a certain position is present in a file. It may result from the sequential composition of an insertion and a deletion, or from the composition of several alter particles, as we will see later.) In the next subsection, we construct composite sequence particles from the prime particles.

Guards One complication arises from the fact that particles of the given applicative algebra A may produce $\perp$. For example, consider the functional composition $g \circ f$ of a substitution g and an alteration f given by an A particle α operating on a position i that g replaces. Obviously the substitution absorbs the alteration, so that $g \circ f = g$, unless $\alpha(x_i) = \perp$ in which case $f(x) = \perp$ and therefore $g \circ f = \perp \neq g$. In this connection, we introduce guards: A *guard* is a unary predicate over A. We write `true` for the guard that always holds and `false` for the unsatisfiable guard.

A. Prime substitution particles

A *prime substitution particle* $f = \mathtt{sub}[p, \sigma, s]$ is given by a natural number p (the *anchor* of f), a sequence σ of guards and a nontrivial replacement sequence s. It is required that

(1) every guard σ_i be satisfiable,
(2) either σ or s be nonempty.

The triple $[p, \sigma, s]$ is the *control* of f.

Given a sufficiently long file x, f replaces the segment $[p, \ldots, p + \#\sigma - 1]$ of x with s provided that x_{p+i} satisfies σ_i for every $i \in \{0, \ldots, \#\sigma - 1\}$. If at least one of these $\#\sigma$ conditions fails, then $f(x) = \perp$. A more detailed description of how f works is given below. The following lemma explains the first of the two requirements.

Lemma 29 *Every prime substitution particle $f = \mathtt{sub}[p, \sigma, s]$ is nontrivial.*

PROOF. Since every σ_i is satisfiable, there is a sequence $y = y_0, \ldots, y_{\#\sigma-1}$ such that every $\sigma_i(y_i)$ is true. Let a be any nontrivial element of A and let x be the sequence of p occurrences of a. Then $f(xy) = xs \neq \perp$. $\square$

The second requirement ensures that no prime substitution particle belongs to the category of position particles defined below.

Arena Define an *arena* to be either a singleton set $\{p\}$ or a nonempty sequence $[p, p+1, \ldots, p+k]$ where p, k are natural numbers. If I and J are arenas, we say that I *is to the left of* J, symbolically $I < J$, if

- either $i < j$ for every $i \in I$ and every $j \in J$,
- or else I is a set $\{i\}$ and J is a sequence and $i \leq j$ for all $j \in J$.

Lemma 30 *The relation $I \leq J \rightleftharpoons I < J \ \vee \ I = J$ is a reflexive partial order on the set of arenas. In other words, the relation is reflexive, transitive and antisymmetric.*

The proof is straightforward.

We define three notions related to a prime substitution $f = \mathtt{sub}[p, \sigma, s]$: the (abstract) *arena* of f, what does it mean that a file x is *long enough* for f, and the *arena of f in file x* that is long enough for f.

Case 1: σ is empty, so that f is an insertion. The (abstract) $\mathtt{arena}(f)$ is the singleton set $\{p\}$. A file x is long enough for f if x is nontrivial and $\#x \geq p$. If x is long enough for f then the arena of f in x is the place number p of file x.

Case 2: σ is nonempty, so that f not an insertion. The (abstract) $\texttt{arena}(f)$ is the sequence $[p, \ldots, p + \#\sigma - 1]$. A file x is long enough for f if x is nontrivial and $\#x \geq p + \#\sigma$. If x is long enough for f then the arena of f in x is the segment of x containing the positions $p, \ldots, p + \#\sigma - 1$ of x.

How a Prime Substitution Works Now we explain in details how a prime substitution $f = \texttt{sub}[p, \sigma, s]$ modifies a file x. If x is not long enough for f or one of the guards in σ fails at x then $f(x) = \bot$. Assume that x is long enough for f and all guards in σ hold.

Case 1: $\#\sigma = 0$. Then f is an insertion. It inserts s at the place p.

Case 2: $\#\sigma > 0$ and s is empty. Then f is a deletion. It removes the arena of f in x. If $x = uvw$ where u is the segment preceding the arena and v is the arena and w is the segment following the arena, then $f(x) = uw$.

Case 3: $\#\sigma > 0$ and s is nonempty. Then f is a proper substitution. It replaces the arena of f in x with s. If $x = uvw$ as above, then $f(x) = usw$.

Adjacency Two prime substitution particles $f = \texttt{sub}[p, \sigma, s]$ and $g = \texttt{sub}[q, \tau, t]$ are *adjacent* if $p + \#\sigma = q$ or $q + \#\tau = p$. In particular, two insertions are adjacent if their arenas coincide.

Lemma 31 *Suppose that prime substitution particles* $f = \texttt{sub}[p, \sigma, s]$ *and* $g = \texttt{sub}[q, \tau, t]$ *are adjacent and* $q \leq p$. *Then* $g \circ f = \texttt{sub}[q, \tau \cdot \sigma, t \cdot s]$.

The proof is obvious.

B. Prime alter particles

We call a nontrivial A particle α *quasi constant* if there is an element $b \neq \bot$ of applicative algebra A such that every $\alpha(a) \in \{b, \bot\}$.

A *prime alter particle* $g = \texttt{alter}[p, \alpha]$ is given by a natural number p, the *anchor position* of g, and by an A particle α, called the *internal particle* of g. It is required that

- $\alpha \neq \bot$ (so that $g \neq \bot$),
- α is not the identity particle (so that no prime alter particle is a prime position particle introduced below),
- α is not quasi constant (so that no prime alter particle belongs to the category of prime substitution particles introduced above).

The pair $[p, \alpha]$ is the *control* of g.

The *arena* of $g = \texttt{alter}[p, \alpha]$ is the singleton sequence $[p]$. A file x is *long enough* for g if $\#x > p$, so that position p exists in x.

Let x be a file. If $x = \bot$ then $g(x) = \bot$. So assume that x is nontrivial. If x is not long enough for g or it is long enough for g but $\alpha(x_p) = \bot$ then $g(x) = \bot$. Otherwise $g(x) = x(p/\alpha(x_p))$, that is $g(x)$ is obtained from x by replacing the element at position p with $\alpha(x_p)$.

C. Prime position particles

A *prime position particle* $h = \texttt{pos}[p]$ is given by a natural number p, the *anchor position* of h. The singleton sequence $[p]$ is the *control* of h.

The *arena* of h is the singleton sequence $[p]$. A file x is *long enough* for h if $\#x > p$, so that position p exists in x.

Let x be a file. If $x = \perp$ or x is not long enough for h, then $h(x) = \perp$. Otherwise $h(x) = x$. Intuitively h requires that the given file is long enough for h.

The definition of prime sequence particles is complete. We took pain to distinguish position particles from substitution particles (be requiring that $\#\sigma + \#s > 0$ for every $\texttt{sub}[p, \sigma, s]$) and from alter particles (by requiring that α is not the identity particle for every $\texttt{alter}[p, \alpha]$). We find it convenient to have position particles as a separate category.

D. Properties of Prime Particles

Let f and g be two prime particles. Define f *is to the left of* g, symbolically $f < g$, if $\texttt{arena}(f) < \texttt{arena}(g)$. To understand the relation better, let us consider three particular cases.

Case 1: f is an insertion particle $\texttt{sub}[p, \epsilon, s]$ and g is a non-insertion prime particle with anchor position q. Then $f < g$ if and only if $p \leq q$.

Case 2: f is a non-insertion prime particle and g is an insertion $\texttt{sub}[q, \epsilon, t]$. Then $f < g$ if and only if $i < q$ for every $i \in \texttt{arena}(f)$.

Case 3: f, g are insertion particles $\texttt{sub}[p, \epsilon, s]$ and $\texttt{sub}[q, \epsilon, t]$ respectively. Then $f < g$ if and only if $p < q$.

Corollary 32 *The relation $f \leq g \rightleftharpoons \texttt{arena}(f) \leq \texttt{arena}(g)$ is a reflexive quasi order on prime particles. In other words, the relation is reflexive and transitive.*

The relation $f \leq g$ is not antisymmetric. Consider two sequence particles f and g with the same arena but with different replacement sequences. Then $f \leq g$ and $g \leq f$ but $f \neq g$.

A sequence $f_1, \ldots, f_n$ of prime particles is a *chain* if $f_i < f_{i+1}$ for every $i \in \{1, \ldots, n-1\}$. By the corollary, every collection of prime particles in which every two particles are ordered forms a chain.

Given a collection C of prime particles that contains a position particle $g = \texttt{pos}[q]$, we say that g is *essential* for C if g is the greatest member of C so that $f < g$ for every $f \in C - \{g\}$.

A chain $f_1, \ldots, f_n$ of prime particles is *normal* if

- for no i, the prime particles f_i and f_{i+1} are adjacent substitution particles, and
- if any f_i is a position particle then $i = n$ (in other words, there are no inessential position particles in the chain).

Lemma 33 *Every chain $f_1, \ldots, f_m$ of prime particles can be transformed into a normal chain $g_1, \ldots, g_n$ such that $f_1 \circ \cdots \circ f_m = g_1 \circ \cdots \circ g_n$.*

PROOF. Repeatedly use Lemma 31 and remove inessential prime position particles. □

We will say that the chain $f_1, \ldots, f_m$ is *normalized* to the chain $g_1, \ldots, g_n$.

Lemma 34 *For every prime particle f, there exists a number $\delta(f)$ such that $\#(fx) - \#x = \delta(f)$ for every file x such that $f(x) \neq \perp$.*

PROOF. If $f = \mathtt{sub}[p, \sigma, s]$ then $\delta(f) = \#s - \#\sigma$. And if f is an alter or position particle then $\delta(f) = 0$. □

The notation $\delta(f)$ in the sense of Lemma 34 will be used in the sequel.

Definition 35 Let f be a prime particle with anchor p and let i be an integer. If $i \leq p$ then $\mathtt{shift}(f, i)$ is the prime particle g that is like f except that the anchor of g is $p - i$. The particle $\mathtt{shift}(f, p)$ will be called the *canonic shift* of f. If $i > p$ then $\mathtt{shift}(f, i) = \perp$.

For example, if $f = \mathtt{alter}[10, \alpha]$ then $\mathtt{shift}(f, 6) = \mathtt{alter}[4, \alpha]$.

Call a file x *appropriate* for a sequence particle f if $f(x) \neq \perp$. Every nontrivial sequence particle has appropriate files.

Corollary 36 *Let f be a prime particle with anchor p and g be the canonic shift of f. Further, let $x = u \cdot v \cdot w$ where $\#u = p$, and $\#v = 0$ if f is an insertion and $\#v = \#(\mathtt{arena}(f))$ otherwise. Then x is appropriate for f if and only if v is appropriate for g. And if x is appropriate for f then $f(x) = u \cdot (gv) \cdot w$.*

3.3 Sequence Particles and their Prime Factors

A *nontrivial sequence particle* f is given by a normal chain of prime particles called the *prime factors* of f. The normal chain of prime factors can be empty in which case f is the identity particle. In addition, the trivial particle $\perp$ is also a sequence particle.

Let f be a nontrivial sequence particle with prime decomposition $[f_1, \ldots, f_n]$. To apply f to a file x, apply all the prime factors of f to x simultaneously. Since the arenas of the prime factors do not overlap, this definition is sound. We explain this in detail. Let $p_1, \ldots, p_n$ be the anchors of $f_1, \ldots, f_n$ respectively and let $p_0 = 0$. Clearly, $p_0 \leq p_1 \leq \cdots \leq p_n$. If $\#x < p_n$ then $fx = \perp$. Assume that $\#x \geq p_n$. The file x divides up into $y_0 \cdot y_1 \cdots y_n$ such that $\#y_i = p_{i+1} - p_i$ for $i < n$. Let $g_1, \ldots, g_n$ be the canonic shifts of $f_1, \ldots, f_n$ respectively. If any $g_i y_i = \perp$ then $fx = \perp$. Otherwise

$$f(x) \rightleftharpoons y_0 \cdot g_1 y_1 \cdot \cdots \cdot g_n y_n$$

We will say that that the prime decomposition $[f_1, \ldots, f_n]$ induces the partition $x = y_0 y_1 \ldots y_n$.

Can this simultaneous application of prime factors be reduced to the usual functional composition? Yes, but only if we apply the prime factors in the right-to-left order.

Corollary 37 *$f(xy) = (fx) \cdot y$ if x is appropriate for f and if $y \neq \perp$.*

PROOF. This is trivial if $f = id$. Otherwise consider a prime decomposition as above and apply Corollary 36 to g_n. $\square$

Corollary 38 *Let $f_1, \ldots, f_n$ be the chain of prime factors of a sequence particle f. Then*

$$f = f_1 \circ f_2 \circ \ \cdots \ \circ f_{n-1} \circ f_n$$

PROOF. Let x be a file appropriate for f and let $x = y_0 y_1 \ldots y_n$ be the partition induced by $[f_1, \ldots, f_n]$. Let g_i be the canonic shift of f_i. By Corollary 36,

$$\begin{aligned}(f_1 \circ \ \cdots \ \circ f_n)(x) &= (f_1 \circ \ \cdots \ \circ f_{n-1})(y_0 \ldots y_{n-1} \cdot g_n y_n) \\ &= (f_1 \circ \ \cdots \ \circ f_{n-2})(y_0 \ldots y_{n-2} \cdot g_{n-1} y_{n-1} \cdot g_n y_n) \\ &= \ldots = f(x) \quad \square\end{aligned}$$

Remark 39 We do not define a distinct family of `overwrite` sequence particles. The effect of overwriting a particular file x by a sequence s can be achieved by using $f = \texttt{sub}[0, \sigma, s]$ where σ is the sequence `true` $\cdots$ `true` of length $\#x$. Note that f is consistent with insertions at the beginning or the end of x.

Lemma 40 *Let f be a nontrivial sequence particle different from id, and let $F = [f_1, \ldots, f_m]$ and $G = [g_1, \ldots, g_n]$ be prime decompositions of f. Then $f_1 = g_1$.*

PROOF. We omit the simple case when f_1 or g_1 is a position particle and assume than neither one is a position particle. Let p, p' be the anchors of f_1, g_1 respectively. Let x range over strings appropriate for f that are long enough so that neither f_1 nor g_1 inserts at the end of x or deletes a suffix of x.

Let j be the maximal number satisfying the following condition $C(j)$: For all x, all $i < j$ and all nontrivial elements a of A, we have that $\#x \geq j$, $\#(fx) \geq j$, $x_i = (fx)_i$, and $x(i/a)$ is appropriate for f. Then $j = p$. Indeed, $C(p)$ holds. It suffices to show that there is an x such that $C(p+1)$ fails. If f_1 is an alter particle then $(fx)_p \neq x_p$ for some x. Otherwise f_1 is some substitution $\texttt{sub}[p, \sigma, s]$. If $\sigma = \epsilon$ then $s \neq \epsilon$ and $(fx)_p = s_0$ independently of x. In this case, there is a nontrivial $a \neq s_0$ such that the sequence $y = x(p/a)$ is inappropriate for f or else $y_p = a \neq s_0 = (fy)_p$. So let $\#\sigma > 0$. If $\sigma_0(a)$ fails for some a, then $x(p/a)$ is not appropriate for f. If $\sigma_0(a)$ holds for all a then $(fx)_p$ does not depend on x_p and therefore x can be chosen so that $x_p \neq (fx)_p$. Similarly $j = p'$ and so $p = p'$.

If f_1 is an alter particle then both x_p and $(fx)_p$ exist but $(fx)_p$ is not constant when x_p varies. Such behavior cannot be achieved by a substitution with anchor p. Therefore g_1 is an alter particle. Furthermore, the dependence of $(fx)_p$ on x_p uniquely determines the inner particle. Similarly, if g_1 is an alter particle then f_1 is the same alter particle. It follows that if one of f_1, g_1 is a substitution then the other is a substitution as well. So assume that f_1, g_1 are substitutions $\texttt{sub}[p, \sigma, s]$ and $\texttt{sub}[p, \tau, t]$ respectively.

Find the greatest k such that if y is obtained from x by changing elements in positions $p, \ldots, p+k-1$ then $fy \in \{fx, \perp\}$. This $k = \#\sigma = \#\tau$. To determine any $\sigma_i(a)$ in case $k > 0$, construct a file $y = x(p + i/a)$ and check whether y is appropriate for f. The same test determines $\tau_i(a)$, and so $\sigma = \tau$. Find the greatest l such that, for every $i \in \{0, \ldots, l-1\}$, the value $(fx)_{p+i}$ does not depend on x. This $l = \#s = \#t$. Further, s and t coincide with the segment $[p, \ldots, p+l-1]$ of fx. $\square$

Lemma 41 *Let f be a nontrivial sequence particle, and let $F = [f_1, \ldots, f_m]$ and $G = [g_1, \ldots, g_n]$ be prime decompositions of f such that $m > 0$ and $n > 0$ and $f_1 = g_1$. Then $f_2 \circ \cdots \circ f_m = g_2 \circ \cdots \circ g_n$.*

PROOF. Let $h = f_1 = g_1$, and $f' = f_2 \circ \cdots \circ f_m$, and $g' = g_2 \circ \cdots \circ g_n$. If an x is appropriate for one of the two particles f', g' then it is appropriate for the other. Indeed suppose that x is appropriate for f'. Recall that different factors of decomposition F operate on disjoint parts of x. Tweak the part of x where h operates so that the resulting y is appropriate for f and therefore for g'. But the different factors of G don't intervene either. x and y look the same to $g_2, \ldots, g_n$. Hence x is appropriate for g'.

If one of the numbers m, n equals 1 then the other equals 1 as well. Indeed suppose that $m = 1$ so that $f' = id$ but $n \geq 2$. Let y be a minimal (by length) string appropriate for h and therefore for f. If there is an i, $1 < i \leq n$, such that the anchor g_i exceeds $\#y$ or the anchor of g_i equals $\#y$ but g_i is not an insertion then y is inappropriate for g_i and therefore inappropriate for f which is impossible. The only possibility remains that $n = 2$, $\texttt{anchor}(g_2) = \#y$ and g_2 is an insertion. In this case, $\#(fx) = \#(h(g_2(y))) > \#(hy) = \#(fx)$ which is impossible.

So we may assume that $m > 1$ and $n > 1$. By contradiction assume that $f' \neq g'$. Then there exists an x appropriate for both particles such that $f'(x) \neq g'(x)$. Split this x into $u \cdot v \cdot w$ where v is the part where f_1 operates. Let $f'' = \texttt{shift}(f_2, \#u + \#v) \circ \cdots \circ \texttt{shift}(f_m, \#u + \#v)$. Construct g'' in the similar way. Clearly $f''(w) \neq g''(w)$. Without loss of generality $hx \neq \perp$; otherwise just replace v with another string appropriate for the canonic shift h' of h. We have $fx = u \cdot (h'v) \cdot (f''w) \neq u \cdot (h'v) \cdot (g''w) = fx$ which is impossible. $\square$

Theorem 42 (Prime Decomposition) *Every nontrivial sequence particle has a unique prime decomposition.*

PROOF. It is easy to see that *id* cannot have a nonempty prime decomposition. So we may restrict attention to nontrivial particles x different from *id*. Given two decompositions of x, apply Lemma 40 to show that the leftmost factors are the same. Then use Lemma 41 to remove the leftmost factors. Repeat this procedure if necessary. $\square$

3.4 Parallel Composition

We define the parallel composition $\mathbf{\Omega}M$ of a multiset M of sequence particles and give an efficient algorithm for computing $\mathbf{\Omega}M$.

A. Parallel Composition of Prime Particles: Definition

Let M be a multiset of prime particles. Recall that prime particles are quasi ordered; see Corollary 32. $\mathbf{\Omega}M = \bot$ if either of the following conditions holds.

C1 M contains particles f, g such that neither $f < g$ nor $g < f$, and f is a substitution, and g either a substitution or an alter particle.

C2 M contains particles $\mathtt{alter}[p, \alpha_1], \ldots, \mathtt{alter}[p, \alpha_n]$ with the same anchor such that $\Omega\{\!\{\alpha_1, \ldots, \alpha_n\}\!\} = \bot$ in A.

Remark 43 Consider a special case when M consists of two substitutions f and g with overlapping arenas. In particular, f and g may be identical. According to C1, M is inconsistent. This reflects the point of view that overlapping substitutions are inconsistent. One may argue for more liberal conflict resolution and, in particular, that, $\{\!\{f, f\}\!\}$ should be consistent with $\mathbf{\Omega}M = f$. This is a legitimate point of view but it needs elaboration. For example, distinct insertions $\mathtt{sub}(p, \epsilon, aa)$ and $\mathtt{sub}(p, \epsilon, aaa)$ may be declared consistent as well.

Suppose that neither C1 nor C2 holds. Then the prime factors of ΩM are obtained by modifying M as follows.

First, fuse together alter particles with the same anchor. This is done separately for each p such that M has an alter particle with anchor p. Form the submultiset $M_p = \{\!\{\mathtt{alter}[p, \alpha_1], \ldots, \mathtt{alter}[p, \alpha_n]\}\!\}$ of the alter particles with anchor p. Compute the parallel composition $\beta_p = \mathbf{\Omega}\{\!\{\alpha_1, \ldots, \alpha_n\}\!\}$ in A.

- If $\beta_p = id$ in A then replace M_p with $\{\!\{\mathtt{pos}[p]\}\!\}$.
- If β_p is quasi constant and b is the nontrivial value of β_p, replace M_p with $\{\!\{\mathtt{sub}[p, \sigma_0, b]\}\!\}$ where $\sigma_0(a)$ is the predicate $\beta_p(a) \neq \bot$.
- Otherwise replace M_p with $\{\!\{\mathtt{alter}[p, \beta_p]\}\!\}$.

Second, remove the superfluous position particles. Keep a position particle only if it is the greatest particle in M; furthermore, keep only one copy of it. Obviously, the remaining particles form a chain.

Third, normalize the chain of remaining particles; see Lemma 33 in this connection. The resulting normal chain is the prime decomposition of ΩM.

B. Parallel Composition of Prime Particles: Efficient Computation

We present an algorithm that requires $n \cdot \log(n)$ primitive operations. We presume that A comes with algorithms for:

- computing the parallel composition,
- determining whether a given particle is trivial,
- determining whether a given particle is the identity,
- determining whether a given particle is quasi constant, and
- determining the unique nontrivial value of the given quasi constant particle.

We consider every application of these algorithms as a single primitive operation. To simplify the exposition, we do not strive to make our algorithm optimal.

Sort the particles in M by the anchors; let M_p be the submultiset of particles with anchor p. In the rest of the algorithm we will be modifying M_p.

Walk through the anchors p and put each particle into one of the following three clusters:

- the cluster A_p of insertions,
- the cluster B_p of alterations and substitutions that are not insertions,
- the cluster C_p of position particles.

If the cardinality of A_p is ≥ 2 then C1 holds; halt and output $\perp$. If the cardinality of B_p is ≥ 2 and B_p contains at least one substitution particle, then C1 holds; halt and output $\perp$. If C_p is nonempty and either p is not the maximal anchor or $B_p \neq \emptyset$, then empty C_p; as a result M_p may become empty in which case discard the anchor p. If the cardinality of C_p is ≥ 2 and p is maximal and $B_p = \emptyset$, then remove all but one member of C_p.

Walk through the anchors p again and, if p is not maximal, compare M_p with M_q where q is the next anchor. If M_p contains a substitution whose arena contains q and M_q contains a non-position particle then C1 holds; halt and output $\perp$.

Walk through the anchors p again. If B_p consists of alterations $\texttt{alter}[p, \alpha_1], \ldots, \texttt{alter}[p, \alpha_n]$, compute β_p as above. It $\beta_p = \perp$ then C2 holds; halt and output $\perp$. If β_p is the identity and p is non-maximal then empty B_p; as a result M_p may become empty in which case discard the anchor p. If β_p is the identity and p is the maximal anchor then empty B_p and put $\texttt{pos}[p]$ into C_p. If β_p is quasi constant with nontrivial value b, replace B_p with $\{\!\{\texttt{sub}[p, \sigma_0, b]\}\!\}$ where $\sigma_0(a)$ is the predicate $\beta_p(a) \neq \perp$. Otherwise replace B_p with $\{\!\{\texttt{alter}[p, \beta_p]\}\!\}$. As a result, every B_p is of cardinality ≤ 1.

Walk through the anchors p again. If $A_p \neq \emptyset$ and B_p consists of a substitution g, then fuse the two substitutions together as in Lemma 31 and put the resulting substitution into B_p instead of g. Also empty A_p.

Walk through the anchors p for the last time. Look for a maximal contiguous segment $p_1, \ldots, p_k$ of anchors satisfying the following conditions: each M_{p_i} contains a substitution and all these k substitutions are adjacent. When you discover such a segment $p_1, \ldots, p_k$, fuse the k substitutions as in Lemma 31.

That completes the description of the algorithm. The correctness of the algorithm is straightforward.

C. Parallel Composition of Arbitrary Sequence Particles: Definition and Efficient Computation

Let M be a multiset of sequence particles. We presume that every nontrivial particle f of M is given by the prime decomposition M_f. If M contains $\perp$ then $\Omega M = \perp$. Otherwise, $\mathbf{\Omega} M$ is the parallel composition of the multiset sum $\sum_{f \in M} M_f$ of prime particles.

It follows that if $f_1, \ldots, f_n$ are the prime factors of a sequence particle f then $f = \Omega\{\!\{f_1, \ldots, f_n\}\!\}$.

3.5 Sequential Composition

The sequential composition of particles f, g is the usual functional composition $g \circ f$. We show that the collection of sequential particles is closed under sequential composition. In the process we give an algorithm that decides whether $g \circ f = \perp$ and finds a prime decomposition of $g \circ f$ if $g \circ f \neq \perp$.

A. Sequential Composition of Prime Particles

We define the sequential composition $g \circ f$ of prime particles f, g. Its prime decomposition contains at most two factors. Recall the notation $\texttt{shift}(f, i)$ (see Definition 35) and recall that with every prime particle f we associate a displacement $\delta(f)$ (see Lemma 34). Let $g' = \texttt{shift}(g, \delta(f))$. We start with two cases where f and g give rise to two distinct factors. In all other cases there will be at most one factor.

Case A: $g < f$, and g, f are not adjacent substitutions, and g is not a position particle. Then $[g, f]$ is the desired prime decomposition of $g \circ f$.

Case B: $g' \neq \perp$ and $f < g'$, and f, g' are not adjacent substitutions, and f is not a position particle. Then $[f, g']$ is the desired decomposition.

The shift is necessary. Consider for example the case when $f = \texttt{sub}[0, \epsilon, a]$ and $g = \texttt{sub}[3, \epsilon, b]$ so that $\delta(f) = 1$ and $g' = \texttt{shift}(g, 1) = \texttt{sub}[2, \epsilon, b]$. If $x = ccc$ then $fx = accc$ and $g(f(x)) = accbc$. The simultaneous application of f and g' to x achieves the same effect.

Assume that neither Case A nor Case B holds.

Case C: $f = \texttt{sub}[p, \sigma, s]$ and $g = \texttt{sub}[q, \tau, t]$. First we consider four simple subcases.

C1: If $g < f$, and g, f are adjacent, then $g \circ f = \texttt{sub}[q, \tau \cdot \sigma, t \cdot s]$.

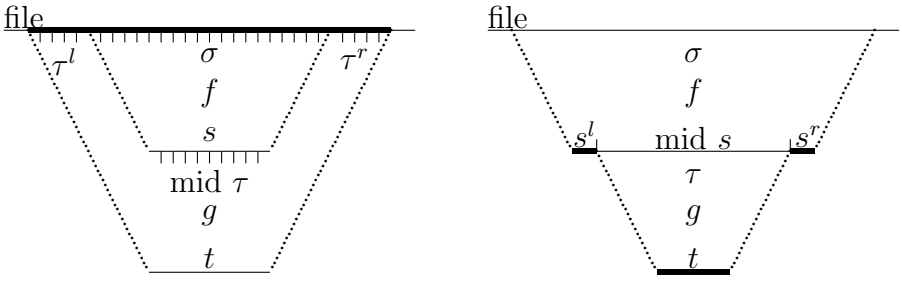
C2: If $g' \neq \perp$ and $f < g'$, and f, g' are adjacent, then $g \circ f = \texttt{sub}[p, \sigma \cdot \tau, s \cdot t]$.

C3: f and g are pure insertions with the same anchor, that is $p = q$ and $\sigma = \tau = \epsilon$. Then $g \circ f = \texttt{sub}[p, \epsilon, t \cdot s]$.

C4: f is an insertion and g the deletion of the string inserted by f, that is $p = q$ and $\sigma = t = \epsilon$ and $\#s = \#\tau > 0$. If any $\tau_i(s_i) = \texttt{false}$ then $g \circ f = \perp$. Otherwise the result depends on p. If $p = 0$, then $g \circ f = id$, and if $p > 0$ then $g \circ f = \texttt{pos}[p - 1]$.

Assume that none of C1–C4 holds. Let x range over files appropriate for f and such that fx is long enough for g. x splits into $u \cdot v \cdot w$ where v is the arena of f in x so that $fx = u \cdot s \cdot w$. Consider a guard τ_i of g (if $\tau \neq \epsilon$). It applies to the element $(fx)_{q+i}$. If the position $q + i$ of $fx = usw$ is located in the u or w part, then $\tau_i((fx)_{q+i})$ is satisfied for some x. But if this position is in the s section then the truth value $\tau_i((fx)_{q+i})$ does not depend on x, only on f and g. If this truth value is $\texttt{false}$ then $g \circ f = \perp$. So assume that $\tau_i((fx)_{q+i}) = \texttt{true}$ for all i such that $q + i$ is in the s section of usw. Then we can fuse f and g into a single substitution particle h; the two pictures illustrate the fusion.

Figure 1



Composing Guards Composing Replacement Sequences

Obviously, the anchor position of the fused substitution h is $\min\{p, q\}$. The guard sequence of h is $\tau^l \cdot \sigma \cdot \tau^r$ where τ^l and τ^r are as follows. $\tau^l = \tau_0 \cdot \ \cdots \ \cdot \tau_{p-q-1}$ which is empty if $q \geq p$. $\tau^r = \tau_{p+\#s-q} \cdot \ \cdots \ \cdot \tau_{\#\tau-1}$ which is empty if $q + \#\tau \leq p + \#s$. The replacement sequence of h is $s^l \cdot t \cdot s^r$ where s^l and s^r are as follows. $s^l = s_0 \cdot \ \cdots \ \cdot s_{q-p-1}$ which is empty if $p \geq q$. $s^r = s_{q+\#\tau-p} \cdot \ \cdots \ \cdot s_{\#s-1}$ is empty if $p + \#s \leq q + \#\tau$.

Case D: $f = \texttt{alter}[p, \alpha]$ and $g = \texttt{alter}[q, \beta]$. As f, g are unordered (neither $f < g$ nor $g < f$), we have $p = q$. Let $\gamma = \beta \circ \alpha$. If $\gamma = \bot$ then $g \circ f = \bot$. If γ is the identity then $g \circ f = \texttt{pos}[p]$. If γ is quasi constant with nontrivial value b then $g \circ f = \texttt{sub}[p, \sigma_0, b]$ where $\sigma_0(a)$ is the predicate $\gamma(a) \neq \bot$. Otherwise $g \circ f = \texttt{alter}[p, \gamma]$.

Case E: $f = \texttt{sub}[p, \sigma, s]$ and $g = \texttt{alter}[q, \beta]$. As f, g are unordered, g transforms an element s_{q-p} of the replacement sequence s into $c \rightleftharpoons \beta(s_{q-p})$. If $c = \bot$, then $g \circ f = \bot$. Otherwise $g \circ f = \texttt{sub}[p, \sigma, s(q - p/c)]$.

Case F: $f = \texttt{alter}[p, \alpha]$ and $g = \texttt{sub}[q, \tau, t]$. As f, g are unordered, $p \in \texttt{arena}(g)$. If $\tau_{p-q} \circ \alpha = \texttt{false}$ then $g \circ f = \bot$. Otherwise $g \circ f = \texttt{sub}[p, \tau(p - q/\tau_{p-q} \circ \alpha), t]$.

Case G: f or g is a position particle.

If f and g are position particles with anchors p and q respectively then $g \circ f = \texttt{pos}[\max\{p, q\}]$.

Suppose that f is a position particle but g is not. As Case A does not apply, g is not to the left of f. Therefore f is inessential and $g \circ f = g$.

Suppose that g is a position particle but f is not. Let x range over files appropriate for f. x splits into $u \cdot v \cdot w$ where $s = \texttt{arena}(f)$ so that $fx = u \cdot v' \cdot w$ for some v'. Let q be the anchor of g. If q is in the u or v sections of x then g is inessential and so $g \circ f = f$. And if q is in the w section then we are in Case B considered above (g is essential and $[f, g']$ form the decomposition).

That completes the analysis of the sequential composition of two prime particles. For future reference we formulate a corollary.

Corollary 44 *Let f, g be two prime particles such that $g \circ f$ is neither trivial nor the identity. Let h_ℓ be the leftmost factor of $g \circ f$. For any prime particle h, if $h < f$ and $h < g$ then $h < h_\ell$.*

This is obvious in every one of the cases above.

B. Sequential Composition in General

Lemma 45 *Consider a list $f_\ell, \ldots, f_1$ of prime particles. If $f_\ell \circ \ \cdots \ \circ f_1 \neq \bot$ then there is a chain $C = [g_m, \ldots, g_1]$ of prime particles with $m \leq \ell$ such that $f_\ell \circ \ \cdots \ \circ f_1 =$*

$g_m \circ \cdots \circ g_1$. *Furthermore, if* $m \geq 1$ *then, for any prime particle* h*, if* h *is to the left of every* f_i *then it is to the left of* g_m*.*

PROOF. We prove the lemma by induction on the length of the list. If the list contains at most one particle then the lemma is obvious. Assume that the lemma holds whenever the list contains at most $\ell \geq 1$ particles. We consider a list $f_{\ell+1}, \ldots, f_1$ and construct the desired chain C.

If $f_\ell \circ \cdots \circ f_1 = \bot$, then $f_{\ell+1} \circ f_\ell \circ \cdots \circ f_\ell = \bot$. So assume $f_\ell \circ \cdots \circ f_1 \neq \bot$. By the induction hypothesis, there is a chain $g_m, \ldots, g_1$ of prime particles such that $f_\ell \circ \cdots \circ f_1 = g_m \circ \cdots \circ g_1$. If $m = 0$, then $C = [f_{\ell+1}]$ and we are done. Otherwise consider $f_{\ell+1} \circ g_m$. By the above analysis, this composition, if not trivial, equals the composition of a chain of at most two prime particles.

- $f_{\ell+1} \circ g_m = \bot$. In this case, $f_{\ell+1} \circ f_\ell \circ \cdots \circ f_1 = (f_{\ell+1} \circ g_m) \circ g_{m-1} \circ \cdots \circ g_1 = \bot$, and we are done.
- $f_{\ell+1} \circ g_m$ is the identity. In this case, $f_{\ell+1} \circ \cdots \circ f_1 = g_{m-1} \circ \cdots \circ g_1$ and thus $[g_{m-1}, \ldots, g_1]$ is the desired chain C.

  If $m \geq 2$ and a prime particle h is to the left of every f_i, then, by the induction hypothesis, it is to the left of g_m and therefore to the left of g_{m-1}.
- $f_{\ell+1} \circ g_m = \bar{g}_m$ for some prime particle $\bar{g}_m$. In this case, $f_{\ell+1} \circ f_\ell \circ \cdots \circ f_1 = f_{\ell+1} \circ g_m \circ \cdots \circ g_1 = \bar{g}_m \circ g_{m-1} \circ \cdots \circ g_1$. As $m \leq \ell$, the induction hypothesis implies that there exists a chain $h_n, \ldots, h_1$ of prime particles such that $\bar{g}_m \circ g_{m-1} \circ \cdots \circ g_1 = h_n \circ \cdots \circ h_1$ and $n \leq m < \ell + 1$.

  If $n \geq 1$ and h is to the left of all f_i then, by the induction hypothesis, h is to the left of g_m. By the Corollary above, h is to the left of $\bar{g}_m$, so that h is to the left of $\bar{g}_m, g_{m-1}, \ldots, g_1$. Applying the induction hypothesis again, we have that h is to the left of h_n.
- $f_{\ell+1} \circ g_m = \bar{g}_{m+1} \circ \bar{g}_m$ where $\bar{g}_{m+1} < \bar{g}_m$ are prime particles. According to the previous analysis, this may may happen only in Cases A and B.

  If $\bar{g}_{m+1}, \bar{g}_m$ arise from Case A, then $f_{\ell+1} = \bar{g}_{m+1} < \bar{g}_m = g_m$. Thus $[\bar{g}_{m+1}, g_m, \ldots, g_1]$ is the desired chain C. For any prime particle h that is to the left of all f_i, we have $h < f_{\ell+1} = \bar{g}_{m+1}$.

  Otherwise $\bar{g}_{m+1}, \bar{g}_m$ arise from Case B. Here, $\bar{g}_{m+1} = g_m < \bar{g}_m$. By induction hypothesis, there is a chain $h_n, \ldots, h_1$ such that $\bar{g}_m \circ g_{m-1} \circ \cdots \circ g_1 = h_n \circ \cdots \circ h_1$ and $n \leq m$. If $n = 0$, the desired $C = [\bar{g}_{m+1}]$. Suppose that $n \geq 1$. Since $\bar{g}_{m+1}$ (that is g_m) is to the left of $\bar{g}_m$ and of all g_i with $i < m$, the induction hypothesis gives us that $\bar{g}_{m+1} < h_n$. The desired chain $C = [\bar{g}_{m+1}, h_n, \ldots, h_1]$. Indeed, $n + 1 \leq m + 1 \leq \ell + 1$ and

$$\begin{aligned}
&\bar{g}_{m+1} \circ h_n \circ h_{n-1} \circ \cdots \circ h_1 = \\
&\bar{g}_{m+1} \circ \bar{g}_m \circ g_{m-1} \circ \cdots \circ g_1 = \\
&f_{\ell+1} \circ g_m \circ g_{m-1} \circ \cdots \circ g_1 = \\
&f_{\ell+1} \circ f_\ell \circ f_{\ell-1} \circ \cdots \circ f_1
\end{aligned}$$

For any prime particle h to the left of all f_i, the induction hypothesis gives $h < g_m =$

$\bar{g}_{m+1}$. $\square$

Theorem 46 *The set of sequence particles is closed under sequential composition.*

PROOF. Let f, g be sequence particles. We show that the operation $g \circ f$ is a sequence particle. If $g \circ f = \perp$ we are done. So assume that $g \circ f \neq \perp$ and so f, g are nontrivial. Let $[f_\ell, \ldots, f_1]$ be the chain of prime factors of f and let $[g_m, \ldots, g_1]$ be the chain of prime factors of g. By Corollary 38, $g \circ f = (\Omega\{\!\{g_1, \ldots, g_m\}\!\}) \circ (\Omega\{\!\{f_\ell, \ldots, f_l\}\!\}) = g_m \circ \cdots \circ g_1 \circ f_\ell \circ \cdots \circ f_1$. By Lemma 45 there is a chain $[h_n, \ldots, h_1]$ such that $g \circ f = h_n \circ \cdots \circ h_1$. Using Lemma 33, this chain is equivalent to a normal chain which is the desired prime decomposition of $g \circ f$. $\square$

3.6 Properties of the Applicative Algebra over Sequences

The applicative algebra *SEQ*(A) has been defined. Let us recapitulate this definition. Recall that A is an arbitrary applicative algebra with element type T that contains at least two nontrivial elements.

(1) The elements of *SEQ*(A) are sequences over T together with the trivial sequence $\perp$. *SEQ*(A) has infinitely many nontrivial elements.
(2) The particles of *SEQ*(A) were introduced in Subsection 3.3. It is easy to see that the requirement AA1 is satisfied. Due to Theorem 46 and the presence of the *id* particle, we have the desired monoid. And we have the trivial particle as well.
(3) The parallel composition was defined in Subsection 3.4. It is easy to see that the requirement AA2 is satisfied.

We establish some properties of *SEQ*(A). Recall the coherence property introduced in Subsection 2.4.

Proposition 47 *Assume that, in A, every consistent multiset is coherent. Then, in SEQ*(A)*, every consistent multiset is coherent.*

PROOF. Let M be a consistent multiset of sequence particles and let x be any sequence such that $fx \neq \perp$ for all $f \in M$. We need to prove that $(\mathbf{\Omega}M)(x) \neq \perp$.

Without loss of generality, M consists of prime particles. If not, let M' be the multiset of all prime factors of all members of M. (Since M is consistent, it does not contain $\perp$, and so each particle in M has a prime decomposition.) Clearly $f(x) \neq \perp$ for all f in M'. By the definition of parallel composition, $\Omega M = \Omega M'$. So it suffices to prove that $(\mathbf{\Omega}M')(x) \neq \perp$.

By the definition of composite particles, $(\mathbf{\Omega}M)(x)$ is the result of simultaneous application of the factors of the prime factors of $\mathbf{\Omega}M$ to x. These factors operate on disjoint parts of x. It suffices to prove that $fx \neq \perp$ for every factor f of $\mathbf{\Omega}M$. The factor construction process is described in Subsection 3.4.

For each p such that M has alter particles with anchor p, you fuse all alter particles $\texttt{alter}[p, \beta_1], \ldots, \texttt{alter}[p, \beta_n]$ of M with the same anchor p. Clearly each $\beta(x_p) \neq \perp$. Let

α be the parallel composition of particles β_i in A, and let f be the result of the fusion. By the hypothesis of the proposition, $\alpha(x_0) \neq \perp$. If $\alpha = id$ then f is a position particle and so $fx \neq \perp$. If α is quasi constant with a nontrivial value b, then $f = \mathtt{sub}[p, \sigma_0, b]$ where $\sigma_0(x_p)$ is the truth value of $\alpha(x_p) \neq \perp$. Again, $fx \neq \perp$. Otherwise, $f = \mathtt{alter}[p, \alpha]$ and again $fx \neq \perp$.

Then you remove superfluous position particles. Of course we have $fx \neq \perp$ for all the remaining particles. These remaining particles form a chain. It remains to normalize the chain. This involves fusing together adjacent substitution particles. But adjacent substitution particles operate on disjoint parts of x. And so we have $fx \neq \perp$ for the result of each instance of fusion. $\square$

Proposition 48 *SEQ(A) is not associative even if A is associative.*

PROOF. Let $f \times g \rightleftharpoons \Omega\{\!\{f, g\}\!\}$ and consider the following example: $h = \mathtt{sub}[0, \mathtt{true}, c]$, $g = \mathtt{sub}[0, \epsilon, b]$, $f = \mathtt{sub}[0, \epsilon, a]$. We have $(h \times g) \times f = \mathtt{sub}[0, \mathtt{true}, bc] \times f = \mathtt{sub}[0, \mathtt{true}, abc]$, but $h \times (g \times f) = h \times \perp = \perp$. $\square$

If A is associative then the only reason that $SEQ(A)$ is not associative is that insertions can be adjacent to proper substitutions or alterations. If we forbid such constellations then $SEQ(A)$ becomes associative.

Proposition 49 *Assume that A is conditionally distributive. Then SEQ(A) is conditionally distributive.*

PROOF. Let M be a consistent multiset of sequence particles, and let $M = M_1 + \cdots + M_n$. We have to prove that $\mathbf{\Omega}M = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \ldots, \mathbf{\Omega}M_n\}\!\}$.

Without loss of generality M consists of prime particles. Indeed, assume that we have proved that if M' is any consistent multiset of prime sequence particles and if $M' = M'_1 + \cdots + M'_n$ then $\mathbf{\Omega}M' = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M'_1, \ldots, \mathbf{\Omega}M'_n\}\!\}$. For every $f \in M$, let M_f be the prime decomposition of f, and let $M' = \sum_{f \in M} M_f$. For every $i = 1, \ldots, n$, let $M'_i = \sum_{f \in M_i} M_f$. By the assumption, $\mathbf{\Omega}M' = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M'_1, \ldots, \mathbf{\Omega}M'_n\}\!\}$. By the definition of the parallel composition of sequence particles, $\mathbf{\Omega}M = \mathbf{\Omega}\{\!\{\mathbf{\Omega}M_1, \ldots, \mathbf{\Omega}M_n\}\!\}$.

The intuitive idea for the rest of the proof is this. Since M is consistent, its particles can be applied simultaneously. Hence the particles of any M_i can be applied simultaneously, and the n groups can be applied simultaneously. The implementation of this idea is straightforward. When you deal with alterations with the same anchor, you have to use the conditional distributivity of A.

Proposition 50 *SEQ(A) is pervasive up (resp. down) if and only if A is so.*

PROOF. A straightforward derivation from the definition of parallel composition. $\square$

4 Application: Labeled Ordered Trees

By *ordered tree* we mean a tree where the children of each node are linearly ordered, and thus form a sequence. (This "horizontal" order on children should be clearly distinguished from the "vertical" partial order of the tree itself.) A *labeled ordered tree* is an ordered tree where each node has a label. What are natural operations on labeled ordered trees? We can think of

- insertion of a labeled ordered subtree,
- deletion of a subtree,
- alteration of a label.

But instead of introducing these and other particle families directly, we use algebra and get a relatively rich world of particles automatically.

For brevity, labeled ordered trees are called *trees* in the rest of this section; no other trees will be considered.

4.1 Construction

Fix an apt applicative algebra L with at least two nontrivial elements. Elements of L will be called labels. Recall the product operation over applicative algebras defined in Subsection 2.7 and consider an operator

$$\Gamma X \rightleftharpoons L \times SEQ(X)$$

where X ranges over applicative algebras. Fixed points of Γ are solutions of the equation $X = \Gamma X$.

We assume that Γ has a fixed point X and analyse X. By the definition of the product of applicative algebras, an arbitrary nontrivial element x of type X has the form (ℓ, s) where ℓ is a label and s is a sequence of elements X. If s is empty, assign rank 0 to x. If $\#s > 0$ and every element s_i of s has been assigned a rank r_i, assign rank $1 + \max_i r_i$ to x. Assign rank 0 to $\bot$. Call an element x *ranked* if it has been assigned a rank.

By the definition of the product of applicative algebras, an arbitrary nontrivial particle f of X has the form $\texttt{alter}[\alpha, g]$ where α is a nontrivial L particle and g is a nontrivial sequence particle. If g does not have any prime alter factors, assign rank 0 to f. In particular the identity particle gets rank 0. If g does have prime alter particles and every prime alter particle h has been assigned a rank r_h, assign rank $1 + \max_h r_h$ to f. Assign rank 0 to $\bot$. Call a particle *ranked* if it has been assigned a rank.

Now consider only ranked elements of X and only ranked particles of X. Furthermore, restrict ranked particles to ranked elements. In the obvious way, every nontrivial element (ℓ, s) of rank r can be viewed as a labeled ordered tree of depth r. In particular (ℓ, ϵ) is a tree with an ℓ-labeled root and no children. A particle $f = \texttt{alter}[\alpha, g]$ operates on a labeled ordered tree $x = (\ell, s)$ in the obvious way: $fx = (\alpha(\ell), gs)$. The parallel composition of particles $\texttt{alter}[\beta_1, g_1], \ldots, \texttt{alter}[\beta_n, g_n]$ is the particle $\texttt{alter}[\alpha, f]$ where $\alpha = \mathbf{\Omega}\{\!\{\beta_1, \ldots, \beta_n\}\!\}$ and $f = \mathbf{\Omega}\{\!\{g_1, \ldots, g_n\}\!\}$. It is easy to see that labeled ordered trees with these particles and this parallel composition of particles form an applicative algebra.

We call this applicative algebra $LOT(L)$, an allusion to *L*abeled *O*rdered *T*rees over L.

Clearly, $LOT(L)$ is a fixed point of Γ and in fact the least fixed point of Γ (with respect to the obvious order).

4.2 Insertions, Deletions, Alterations

We show that the operations of insertion, deletion and label alteration informally introduced in the preamble of this section give rise to legitimate tree particles. We do not claim that every tree particle is an insertion, deletion or alteration; it will be easy to see that there are more tree particles.

We start by defining an auxiliary function $\mathtt{node}(x, s)$ that, given a tree x and a sequence s of natural numbers produces the subtree of x at position s. If s is empty, then $\mathtt{node}(x, s)$ is x. If $s = [i]$ then $\mathtt{node}(x, s)$ is the child number i of x (where the leftmost child is child number 0) provided that x has at least $i + 1$ children; otherwise $\mathtt{node}(x, s) = \bot$. If $s = [i, j]$ and $\mathtt{node}(x, [i]) \neq \bot$ and $\mathtt{node}(x, [i])$ has a child y of number j then $\mathtt{node}(x, s) = y$; otherwise $\mathtt{node}(x, s) = \bot$. And so on.

In the rest of this subsection, we describe the three natural operations over trees mentioned above – insertion, deletion and label alteration – more precisely and check that they are legitimate particles of $LOT(L)$.

A. Insertion

The insertion operation $f = \mathtt{insert}[s, y]$ takes two parameters: a nonempty sequence s of natural numbers and a tree y. Since s is nonempty it has the form $t \cdot [p]$. Let x be a nontrivial tree. If $\mathtt{node}(x, t) = \bot$ or else $\mathtt{node}(x, t) \neq \bot$ and $p > 0$ and $\mathtt{node}(x, t)$ has less than p children then $f(x) = \bot$. So assume that $\mathtt{node}(x, t)$ is a nontrivial tree with at least p children. If $\mathtt{node}(x, t)$ has exactly p children so that the last child is number $p-1$, then attach y as the p^{th} child of $\mathtt{node}(x, t)$. If $\mathtt{node}(x, t)$ has more than p children, then insert y between the child number $p - 1$ and the child number p of $\mathtt{node}(x, t)$ so that y becomes the child number p of $\mathtt{node}(t, x)$.

By the definition of $LOT(L)$, the only particles of $LOT(L)$ are alter particles. We illustrate that operations $\mathtt{insert}[s, y]$ are legitimate particles. Clearly

$$\mathtt{insert}[[q], y] = \mathtt{alter}[id,\ \mathtt{sub}[q,\ \epsilon,\ y]].$$

Let g be a sequence particle $\mathtt{alter}[p,\ \mathtt{insert}[[q], y]]$. Given a sequence of at least $p + 1$ trees, g applies the tree particle $\mathtt{insert}[[q], y]$ to the p^{th} tree (and produces $\bot$ if the sequence is too short). Then

$$\mathtt{insert}[[p, q], y] = \mathtt{alter}[id,\ g].$$

B. Deletion

The deletion operation $f = \mathtt{delete}[s]$ takes one parameter: a nonempty sequence s of natural numbers. Since s is nonempty it has the form $t \cdot [p]$. Let x be a nontrivial tree. If

$\texttt{node}(x,t) = \perp$ or else $\texttt{node}(x,t)$ is nontrivial but does not have the child number p then $f(x) = \perp$. Otherwise $f(x)$ is obtained from x by deleting the p^{th} child of $\texttt{node}(x,t)$.

We illustrate that the delete operations are legitimate particles. Clearly

$$\texttt{delete}[[q]] = \texttt{alter}[id, \texttt{sub}[q, \texttt{true}, \epsilon]].$$

Let g be a sequence particle $\texttt{alter}[p, \texttt{delete}[[q]]]$. Given a sequence of at least $p+1$ trees, g applies the tree particle $\texttt{delete}[[q]]$ to the p^{th} tree (and produces $\perp$ if the sequence is too short). Then

$$\texttt{delete}[[p,q]] = \texttt{alter}[id, g].$$

C. Label Alteration

The label operation $f = \texttt{la}[s, \alpha]$ takes two parameters: a possibly empty sequence s of natural numbers and a nontrivial label particle $\alpha \neq id$. Let x be a nontrivial tree. If $\texttt{node}(x,s) = \perp$ or if $\texttt{node}(x,s)$ is nontrivial with some label ℓ but $\alpha(\ell) = \perp$ then $f(x) = \perp$. Otherwise $f(x)$ is obtained from x by replacing the label ℓ of $\texttt{node}(x,s)$ with $\alpha(\ell)$.

We illustrate that label operations are legitimate particles. Clearly

$$\texttt{la}[\epsilon, \alpha] = \texttt{alter}[\alpha, id].$$

Let f be a sequence particle $\texttt{alter}[q, \texttt{alter}[\alpha, id]]$. Given a sequence of at least $q+1$ trees, f applies the tree particle $\texttt{la}[\epsilon, \alpha]$ to the q^{th} tree (and produces $\perp$ if the sequence is too short). Then

$$\texttt{la}[[q], \alpha] = \texttt{alter}[id, f].$$

Let g be a sequence particle $\texttt{alter}[p, \texttt{la}[[q], \alpha]]$. Given a sequence of at least $p+1$ trees, g applies the tree particle $\texttt{la}[[q], \alpha]$ to the p^{th} tree (and produces $\perp$ if the sequence is too short). Then

$$\texttt{la}[[p,q], \alpha] = \texttt{alter}[id, g].$$

References

[1] Eric Allman, "An Introduction to the Source Code Control System," University of California at Berkeley, Working paper, 1980.

[2] The ASM academic web page, `http://www.eecs.umich.edu/gasm`, maintained by Jim Huggins.

[3] Brian Berliner, "CVS II: Parallelizing Software Development," Proceedings of the Winter 1990 USENIX Association Conference, 1990.

[4] G. Birkhoff, "Lattice Theory", Colloquium Publications XXV, American Mathematical Society, 3rd edition, 1967.

[5] Egon Börger, and Robert Stärk, "Abstract State Machines: A Method for High-Level System Design and Analysis", Springer-Verlag, Berlin, Heidelberg, 2003.

[6] C. A. Ellis, and S. J. Gibbs, "Concurrency Control in Groupware Systems," Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 399–407, May 1989.

[7] The web page of the group on Foundations of Software Engineering at Microsoft Research, `http://research.microsoft.com/fse/`.

[8] GNU, "Comparing and Merging Files", Edition 2.8.1, dated 5 April 2002, `http://www.gnu.org/software/diffutils/manual/html_node/index.html`.

[9] Yuri Gurevich, "Evolving Algebra 1993: Lipari Guide," *Specification and Validation Methods*, E. Börger, ed., Oxford Uni. Press (1995), pp. 9–36.

[10] Yuri Gurevich, and Nikolai Tillmann, "Partial Updates: Exploration," *Springer Journal of Universal Computer Science* 7:11 (2001), 917–951.

[11] Microsoft Word, "Comparing and Merging Documents," `http://office.microsoft.com/assistance/category.aspx?CategoryID=CH063551081033`

[12] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhauser, "An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors," Proceedings of the ACM Conference on Computer Supported Cooperative Work, pp. 288–297, 1996.

[13] M. Suleiman, M. Cart and J. Ferrié, "Serialization of Concurrent Operations in Distributed Collaborative Environment," Proceedings of the ACM International Conference on Supporting Group Work (GROUP'97), Phoenix, pp. 435–445, November 1997.

[14] C. Sun, C. Ellis, "Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements," Proceedings of the ACM Conference on Computer-Supported Cooperative Work [2], pp. 59–68, 1998.

[15] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving Convergence, Causality-Preservation, and Intention-Preservation in Real-Time Cooperative Editing Systems," ACM Transactions on Computer-Human Interaction, Vol. 5, No. 1, pp. 63–108, 1998.

[16] Walter F. Tichy, "Design, Implementation and Evaluation of a Revision Control System," Proceedings of 6th International Conference on Software Engineering, IEEE, Tokyo, September 1982.