A FORMALIZATION OF DARCS PATCH THEORY USING INVERSE SEMIGROUPS

JUDAH JACOBSON

Abstract. Darcs is a distributed version control system with a flexible repository model. An abstract "theory of patches" determines how changes from multiple sources are combined and transformed while maintaining a consistent history. Previous formalizations of the semantics of Darcs have generally relied on definitions and assumptions chosen specifically to model that system.

We present a formalization of patch theory which builds upon established mathematical concepts. In particular, we model patch effects as elements of *inverse semigroups*, which generalize the properties of partial injective functions. We demonstrate that our approach provides a useful framework for discussing and proving results about the operations performed by Darcs. We also give new insight into *conflictors*, which are used to merge incompatible changes and can affect reliability and performance of the software.

1. Patch theory

1.1. **Version control systems.** A version control system manages changes to a repository of files. More traditional software such as Subversion [20] and Perforce [14] synchronize changes from different users by means of a central repository. In contrast, distributed revision control systems (DVCS) allow each user to record changes in their own copy of the repository. Users can asynchronously apply changes from other repositories to their own.

Some care is needed when merging changes between repositories. For example, suppose that Alice and Bob each start with identical copies of the repository. Alice makes the changes

(A) insert `'foo'` as a new line at the beginning of file F
(B) rename file F to G

while Bob independently makes the change

(C) change line five of F from `'bar'` to `'baz'`

Then the following is a valid merger of their changes:

(A) insert `'foo'` as a new line at the beginning of file F
(B) rename file F to G
(C′) change line six of file G from `'bar'` to `'baz'`

Notice how the line number and filename of (C) must be altered in order to obtain a consistent history.

Darcs [3] is a DVCS which builds its merging algorithms on a "theory of patches." Patch theory [2, 5, 11, 18] is an attempt to define patches abstractly and to provide provably correct merging algorithms. The authors of Darcs assert that their approach makes the software simpler and more robust.

This paper presents a new formalization of patch theory which builds upon established mathematical concepts. Specifically, we model patch effects as elements of an *inverse semigroup*. We describe *commutation*, which reorders patches while preserving their original intent, and show how it enables Darcs to merge changes from different sources. We prove that certain operations performed by Darcs are *sensible*, i.e. produce patch sequences which make consistent changes to the repository state. We furthermore use our model to discuss two open areas of research: *permutivity*, which affects synchronization between more than two repositories; and *conflictors*, which resolve incompatible changes from different users.

1.2. **Modelling patches.** A "patch" in Darcs denotes a change in the repository state. For example:

- Add or remove an empty file
- Move (i.e., rename) a file
- Replace a contiguous sequence of lines with another (which may be of different length)
- Replace all instances of one word in the file with another

Letting R represent the set of repository states, we model changes as partial one-to-one functions $f : D \subseteq R \to R$. Two partial functions f and g may be composed on a restricted set; i.e., $(fg)(x) = g(f(x))$ when

$$x \in \mathrm{Dom}(fg) = f^{-1}\left(\mathrm{Ran}(f) \cap \mathrm{Dom}(g)\right).$$

Our choice for order of composition differs from the usual mathematical notation, but will allow us to better match the standard formulations of patch theory. Also note that the above composition may result in the "zero" function with empty domain and range.

Every partial one-to-one f has a unique inverse $f^{-1} : \mathrm{Ran}(f) \to \mathrm{Dom}(f)$ such that $f^{-1}(f(x)) = x$ on $\mathrm{Dom}(f)$ and $f(f^{-1}(x)) = x$ on $\mathrm{Ran}(f)$. For example:

- The function "add a file F" has domain $\{F$ does not exist$\}$ and range $\{F$ exists and is empty$\}$.
- Its inverse "remove the file F" has domain $\{F$ exists and is empty$\}$ and range $\{F$ does not exist$\}$.
- The function "rename file F to G" has domain $\{F$ exists and G does not exist$\}$ and range $\{F$ does not exist and G exists$\}$.
- The function "insert one line `'foobar'` at the start of F" has domain $\{F$ exists$\}$ and range $\{F$ exists and its first line is `'foobar'`$\}$.

The following definition [7, 8] generalizes the above concepts of composition and inversion:

Definition 1.1. A set X with a binary multiplication operation is called an *inverse semigroup* if:

- Multiplication is associative; i.e., $x(yz) = (xy)z$ for all $x, y, z \in X$.
- Every $x \in X$ has a unique inverse x^{-1} such that $xx^{-1}x = x$ and $x^{-1}xx^{-1} = x^{-1}$.

Inverses in the above definition follow some of the same laws as in groups; for example, it can be shown that $(xy)^{-1} = y^{-1}x^{-1}$. However, in general xx^{-1} is not the identity element (if one such even exists in X). It is idempotent, though, since $(xx^{-1})(xx^{-1}) = x(x^{-1}xx^{-1}) = xx^{-1}$.

Definition 1.2. An *idempotent* of a semigroup is an element y with $yy = y$.

Idempotents play an important role in the theory of inverse semigroups. For example, if f and g are partial one-to-one functions then $ff^{-1} = \mathrm{id}_{\mathrm{Dom}(f)}$ and $ff^{-1}g$ is the restriction $g|_{\mathrm{Dom}(g)\cap\mathrm{Dom}(f)}$; similarly $f^{-1}f = \mathrm{id}_{\mathrm{Ran}(f)}$.

We are led to a natural partial ordering: $g \leq f$ if $\mathrm{Dom}(g) \subseteq \mathrm{Dom}(f)$ and $g = f$ on $\mathrm{Dom}(g)$. That ordering generalizes to arbitrary inverse semigroups:

Definition 1.3. Let x, y be elements of an inverse semigroup X. Then $x \leq y$ if $x = ay$ for some idempotent a.

It can be shown that this relation is equivalent to letting $x \leq y$ if $x = ya$ for some idempotent a. Furthermore, this relation has the following properties:

- $x \leq y,\ y \leq x \implies x = y$ (antisymmetry)
- $x \leq y,\ y \leq z \implies x \leq z$ (transitivity)
- $w \leq x,\ y \leq z \implies wy \leq xz$
- $w \leq x \iff w^{-1} \leq x^{-1}$.

Finally, we can generalize the partial function with empty domain:

Definition 1.4. A semigroup X has a zero element (denoted "0") if $0x = x0 = 0$ for any $x \in X$.

The following properties can be shown for any $x, y \in X$:

- $x \geq 0$.
- If $xy \neq 0$ then $x, y \neq 0$.
- $x \neq 0$ iff $x^{-1} \neq 0$.

1.3. **Patch systems.** The patches in Darcs are concretely representable units of change. Patches also contain metatdata such as a description, a unique identifier or the name of the user who originally recorded it. Since metadata is relevant to some merging algorithms, we will model the concepts of patches and changes as distinct mathematical objects.

We choose the following model:

Definition 1.5. Call a set $\mathcal{P}$ a *patch set* if it has an inversion operation satisfying $(p^{-1})^{-1} = p$ for all $p \in \mathcal{P}$.

Definition 1.6. Let $\mathcal{P}$ be a patch set, S be an inverse semigroup, and $\mathcal{E} : \mathcal{P} \to S$ a function satisfying $\mathcal{E}(p^{-1}) = \mathcal{E}(p)^{-1}$ for all $p \in \mathcal{P}$. Then we say that $(\mathcal{P}, S, \mathcal{E})$ is a *patch system*.

Informally, $\mathcal{E}$ assigns an *effect*, or change of the shared state, to each patch in $\mathcal{P}$.

The history of a repository contains a list of consecutive changes; thus we will also consider sequences of patches.

Definition 1.7. Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system. Then $\mathcal{P}^*$ is the set of all *patch sequences* $\mathbf{p} = (p_1, p_2, \ldots, p_n)$ where each $p_i \in \mathcal{P}$. We define inversion on $\mathcal{P}^*$ by

$$(p_1, p_2, \ldots, p_n)^{-1} = ({p_n}^{-1}, \ldots, {p_2}^{-1}, {p_1}^{-1}).$$

Furthermore, we extend $\mathcal{E}$ to the map $\mathcal{E}^* : \mathcal{P}^* \to S$ by

$$\mathcal{E}^*((p_1, \ldots, p_n)) = \mathcal{E}(p_1) \ldots \mathcal{E}(p_n).$$

When convenient, we will denote a sequence (p) which consists of only one patch as just p. We also define multiplication in $\mathcal{P}^*$ to be sequence concatenation.

The following proposition follows trivially from the definitions:

Proposition 1.1. *Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system. Then $(\mathcal{P}^*, S, \mathcal{E}^*)$ is also a patch system.*

It can also be extended to a nice algebraic formulation:

Proposition 1.2. *Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system. Then $\mathcal{P}^*$ is a* semigroup with involution*; namely the inversion operation satisfies the laws $(\mathbf{p}^{-1})^{-1} = \mathbf{p}$ and $(\mathbf{pq})^{-1} = \mathbf{q}^{-1}\mathbf{p}^{-1}$.*

Furthermore, $\mathcal{E}^*$ is a semigroup homomorphism which preserves involution; i.e.,

$$\mathcal{E}^*(\mathbf{pq}) = \mathcal{E}^*(\mathbf{p})\mathcal{E}^*(\mathbf{q}) \text{ and } \mathcal{E}^*(\mathbf{p}^{-1}) = \mathcal{E}^*(\mathbf{p})^{-1}.$$

The definitions so far permit arbitrary sequences of patches; however, in practice some combinations are not reasonable. For example, a file cannot be edited before it has been added to the repository. We model that concept as follows:

Definition 1.8. Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system. Then we say that a patch $p \in \mathcal{P}$ is *sensible* if $\mathcal{E}(p) \neq 0$.

As an example use of the theory, we prove that it is always sensible to apply the inverse of the most recent patch:

Proposition 1.3. *Let S be an inverse semigroup, and $x, y \in S$. If xy is nonzero, then so is xyy^{-1}.*

Proof. Suppose instead that $0 = xyy^{-1}$. Then

$$0 = 0y = xyy^{-1}y = xy,$$

contradicting the assumption that xy was nonzero. □

Corollary 1.4. *If the patch sequence $\mathbf{pq}$ is sensible, then so is $\mathbf{pqq}^{-1}$.*

2. Commutation

2.1. **Description.** Suppose that Alice and Bob each start with the same history of changes. Alice, in her copy of the repository, records the patch $p =$"move file F to G;" whereas Bob records $q =$"insert `'foobar'` at the start of F." Care must be taken when merging their changes into a single consistent history; for example, the sequence pq is not sensible. Instead, the two changes may be sensibly combined into either qp or pr, where $r =$"insert `'foobar'` at the start of G."

Darcs uses merging algorithms which are built on the concept of patch commutation. Informally, a pair of patches (p, q) commutes when we can find another pair (q', p'), with p' and q' having the same "meaning" as p and q respectively, such that $pq \equiv q'p'$. We write this relation as $(p, q) \leftrightarrow (q', p')$. (If $pq = qp$, we will say that p and q *trivially commute*.)

For example, the patches in the above example are related by commutation, namely $(q, p) \leftrightarrow (p, r)$. Not every pair of patches may be commuted; for example, adding a file and then modifying it only makes sense in the original order. In that case we say that the latter patch *depends* on the former.

In addition to providing a flexible approach to merging (which we will revisit in a later section), commutation enables "cherry-picking" from a patch sequence. For

example, suppose Alice has the patch sequence $\mathbf{p}$, Bob has $\mathbf{p}qr$, and Alice wishes to synchronize with Bob's history but does not want patch q. She cannot simply pull r to form $\mathbf{p}r$; in particular, there is no guarantee that sequence is sensible. Instead, she first commutes the last two patches of Bob's history, resulting in $\mathbf{p}r'q'$; this new sequence is sensible since it has the same effect as Bob's. Then, she adds r' to her history to obtain the also-sensible (as well as semantically correct) $\mathbf{p}r'$.

Previous formulations of Darcs patch theory (e.g., [11, 19]) have started by assuming that commutation satisfies several useful properties. The following definition starts integrating those requirements into our model.

Definition 2.1. Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system. Let $\leftrightarrow$ be a relation on pairs of elements of $\mathcal{P}$. Then we call $\leftrightarrow$ a *commutation relation* if all of the following hold:

- Effect-preserving: If $(p, q) \leftrightarrow (r, s)$, then $\mathcal{E}(pq) = \mathcal{E}(rs)$.
- Symmetric: If $(p, q) \leftrightarrow (r, s)$, then $(r, s) \leftrightarrow (p, q)$.
- Rotating: If $(p, q) \leftrightarrow (r, s)$ then $\left(r^{-1}, p\right) \leftrightarrow \left(s, q^{-1}\right)$.

Informally, the rotating property asserts that we may rotate the following commutative diagram clockwise:

$$\begin{array}{ccc} A & \xrightarrow{p} & B \\ \downarrow{\scriptstyle r} & & \downarrow{\scriptstyle q} \\ C & \xrightarrow{s} & D \end{array} \quad \Longrightarrow \quad \begin{array}{ccc} C & \xrightarrow{r^{-1}} & A \\ \downarrow{\scriptstyle s} & & \downarrow{\scriptstyle p} \\ D & \xrightarrow{q^{-1}} & B \end{array}$$

From repeated application of the symmetric and rotating properties, it follows that the diagram stays correct under arbitrary flips or rotations. In particular, by applying the symmetric property twice we find that $(p, q) \leftrightarrow (r, s)$ if and only if $(r, s) \leftrightarrow (p, q)$; and by applying the rotating property four times we find that

$$\begin{aligned} (p, q) \leftrightarrow (r, s) &\iff \left(r^{-1}, p\right) \leftrightarrow \left(s, q^{-1}\right) \\ &\iff \left(s^{-1}, r^{-1}\right) \leftrightarrow \left(q^{-1}, p^{-1}\right) \\ &\iff \left(q, s^{-1}\right) \leftrightarrow \left(p^{-1}, r\right). \end{aligned}$$

Definition 2.2. Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system with commutation relation $\leftrightarrow$. We say that $\leftrightarrow$ is *uniquely commuting* if for each p and q in $\mathcal{P}$ there is at most one pair (r, s) with $(p, q) \leftrightarrow (r, s)$.

If that uniqueness property is satisfied, then we can consider commutation as an operation on patches, rather than a relation. This view helps derive properties of the algorithms of Darcs. For example, uniqueness plus symmetry implies that commuting two patches twice must result in the original pair.

2.2. **Commutation of sequences.**

Most of Darcs' commands transform a history of changes rather than just a single patch. As a result, we also consider commutation relations on patch sequences.

For example, let $\mathbf{p} = (p_0, p_1)$ and $\mathbf{q} = (q_0, q_1)$. By commuting adjacent primitive patches, we may reorder $\mathbf{pq}$ into $\mathbf{q}''\mathbf{p}''$ as follows:

$$\begin{aligned}
\mathbf{pq} = (p_0, p_1, q_0, q_1) &\leftrightarrow (p_0, q_0', p_1', q_1) && \text{if } (p_1, q_0) \leftrightarrow (q_0', p_1') \\
&\leftrightarrow (q_0'', p_0', p_1', q_1) && \text{if } (p_0, q_0') \leftrightarrow (q_0'', p_0') \\
&\leftrightarrow (q_0'', p_0', q_1', p_1'') && \text{if } (p_1', q_1) \leftrightarrow (q_1', p_1'') \\
&\leftrightarrow (q_0'', q_1'', p_0'', p_1'') = \mathbf{q}''\mathbf{p}'' && \text{if } (p_0', q_1') \leftrightarrow (q_1'', p_0'')
\end{aligned}$$

We say that $(\mathbf{p}, \mathbf{q}) \leftrightarrow^* (\mathbf{q}'', \mathbf{p}'')$ if each pair of primitive patches commutes as above.

It is possible to express the above relationships succinctly as a commutative diagram, as in [16]:

$$\begin{array}{ccccc}
 & \xrightarrow{p_0} & & \xrightarrow{p_1} & \\
\downarrow{\scriptstyle q_0''} & & \downarrow{\scriptstyle q_0'} & & \downarrow{\scriptstyle q_0} \\
 & \xrightarrow{p_0'} & & \xrightarrow{p_1'} & \\
\downarrow{\scriptstyle q_1''} & & \downarrow{\scriptstyle q_1'} & & \downarrow{\scriptstyle q_1} \\
 & \xrightarrow{p_0''} & & \xrightarrow{p_1''} &
\end{array} \tag{1}$$

The definitions from the previous section apply naturally to patch sequences:

Proposition 2.1. *Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system with commutation relation $\leftrightarrow$. Then we can construct a commutation relation $\leftrightarrow^*$ for the patch system $(\mathcal{P}^*, S, \mathcal{E}^*)$.*

Furthermore, if $\leftrightarrow$ is uniquely commuting, then so is $\leftrightarrow^*$.

A complete formal construction and proof requires some tedious notation. Thus we will give an outline of the necessary steps, and defer the details to a separate paper.

Proof. Assume that we are given a patch system with an associated commutation relation $\leftrightarrow$. Then we say that two patch sequences $\mathbf{p} = (p_0, \dots, p_n)$ and $\mathbf{q} = (q_0, \dots, q_m)$ commute if there exist doubly-indexed sequences of intermediate patches $p_i^{(k)}$ and $q_j^{(l)}$ filling in a diagram similar to (1), such that each intermediate "box" of commutation holds:

$$\begin{array}{ccc}
 & \xrightarrow{p_{m-i}^{(j)}} & \\
\downarrow{\scriptstyle q_j^{(i+1)}} & & \downarrow{\scriptstyle q_i^{(j)}} \\
 & \xrightarrow{p_{j+1}^{(m-i)}} &
\end{array}$$

Now, by rotating (1) ninety degrees we find that rotation for $\leftrightarrow^*$ follows directly from rotation for $\leftrightarrow$. Similarly, flipping (1) along the diagonal demonstrates symmetry of $\leftrightarrow^*$ from the symmetry of $\leftrightarrow$.

For effect-preservation, we consider paths through the arrows of (1) which start at the top left corner and finish at the bottom right. The path which takes the top and right sides of the diagram has effect $\mathcal{E}^*(\mathbf{pq})$. We can use effect-preservation of $\leftrightarrow$ to show by induction that the same effect must result from any path; and in particular, that includes the path taking the left and bottom sides.

Finally, assume that $\leftrightarrow$ is uniquely commuting. Then for each intermediate box of commutation, the top and right sides specify unique values for the left and bottom

sides. But specifying $\mathbf{p}$ and $\mathbf{q}$ specifies the top and right sides of (1), so we can use induction to show that $\mathbf{p}$ and $\mathbf{q}$ also specify unique values at each intermediate step. □

3. Merging

In this section we will provide a simplified description of how Darcs uses commutation to merge patches from different authors.

Assume that Alice has history $\mathbf{cp}$ and Bob has history $\mathbf{cq}$. They synchronize repositories by adding the patches from the other user to the end of their own history. Namely:

- Alice gets $\mathbf{cpr}$, where $\mathbf{r}$ has the same "meaning" as $\mathbf{q}$ but makes sense when applied after $\mathbf{p}$;
- Bob gets $\mathbf{cqs}$, where $\mathbf{s}$ has same "meaning" as $\mathbf{p}$ but can be applied after $\mathbf{q}$.

Since the two repositories ought to be consistent, we expect that $\mathcal{E}^*(\mathbf{pr})$ and $\mathcal{E}^*(\mathbf{qs})$ should be equal. Thus, assuming that this patch "meaning" is the same concept as in commutation, we merge $\mathbf{p}$ and $\mathbf{q}$ by looking for sequences $\mathbf{r}$ and $\mathbf{s}$ with $(\mathbf{p}, \mathbf{r}) \leftrightarrow (\mathbf{q}, \mathbf{s})$.

Furthermore, by the symmetric and rotating properties we have

$$\begin{aligned}(\mathbf{p}, \mathbf{r}) \leftrightarrow (\mathbf{q}, \mathbf{s}) &\iff \left(\mathbf{q}^{-1}, \mathbf{p}\right) \leftrightarrow \left(\mathbf{s}, \mathbf{r}^{-1}\right) \\ \iff (\mathbf{q}, \mathbf{s}) \leftrightarrow (\mathbf{p}, \mathbf{r}) &\iff \left(\mathbf{p}^{-1}, \mathbf{q}\right) \leftrightarrow \left(\mathbf{r}, \mathbf{s}^{-1}\right).\end{aligned}$$

Therefore we can compute $\mathbf{r}$ and $\mathbf{s}$ by commuting either $\left(\mathbf{p}^{-1}, \mathbf{q}\right)$ or $\left(\mathbf{q}^{-1}, \mathbf{p}\right)$. If $\mathbf{p}^{-1}$ and $\mathbf{q}$ cannot be commuted, then we say that $\mathbf{p}$ and $\mathbf{q}$ *conflict*; we will discuss how Darcs handles that case in Section 5.

Another way to understand the relevance of commuting $\left(\mathbf{p}^{-1}, \mathbf{q}\right)$ is the following. Assuming that $\mathbf{cp}$ and $\mathbf{cq}$ are sensible, we might hope that $\mathbf{cpp}^{-1}\mathbf{q}$ is also sensible. If so, then $\left(\mathbf{p}^{-1}, \mathbf{q}\right) \leftrightarrow \left(\mathbf{r}, \mathbf{s}^{-1}\right)$ means that we can replace $\mathbf{p}^{-1}\mathbf{q}$ with $\mathbf{rs}^{-1}$ to form $\mathbf{cprs}^{-1}$. That sequence is still sensible since $\leftrightarrow$ is effect-preserving; thus its subsequence $\mathbf{cpr}$ must also be sensible.

However, in the general setting it is not enough to assume that $\mathbf{cp}$ and $\mathbf{cq}$ are sensible. For example, let

- $p =$"change the first line of file F from `'foo'` to `'bar'`"
- $q =$"change the second line of file F from `'foo'` to `'baz'`"
- $\mathcal{E}^*(\mathbf{c})$ be some partial function whose range is {Either the first or second line of F is `'foo'`, but not both}.

Then $\mathbf{c}p$ and $\mathbf{c}q$ are both sensible; and their merger ought to be $\mathbf{c}pq$ since $(p, q) \leftrightarrow (q, p)$. But $\mathrm{Dom}(pq)$ is {Both the first and second lines of F are `'foo'`}, so $\mathbf{c}pq$ is not sensible.

As a result, different assumptions are necessary in order to prove that a merged history is sensible. We start with the following definition:

Definition 3.1. Let X be an inverse semigroup. Then $x \in X$ is called *minimal* if there is no nonzero element strictly less than x.

For example, any partial function whose domain consists of a single element is minimal. (The mathematical literature (e.g., [8]) refers to minimal idempotents as "primitive," but that name is already part of the terminology of Darcs.)

Definition 3.2. Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system, and let e be an idempotent in S. We say that $p \in \mathcal{P}$ is *e-sensible* if $e\mathcal{E}(p)$ is nonzero.

In particular, let R be the state of a repository with no files or folders, and let e be the identity function whose domain is the singleton set $\{R\}$. Since every Darcs repository is initialized to R, its full history must be an e-sensible patch sequence. The same holds for any subsequence which starts at the beginning of the history.

To prove that a merger is e-sensible, we will need the following lemma.

Lemma 3.1. *Let X be an inverse semigroup. If $x, e \in X$ and e is a minimal idempotent, then ex is also minimal.*

Proof. Fix $x, z, e \in X$, where e is a minimal idempotent and $z \neq 0$. If $z \leq ex$ then $z = exy$ for some idempotent $y \in X$; we will show that $exy = ex$.

It is a basic fact of inverse semigroups that idempotents trivially commute [7, 8]. In particular, $x^{-1}xy = yx^{-1}x$, so

$$0 \neq exy = exx^{-1}xy = exyx^{-1}x.$$

Therefore $exyx^{-1}$ must also be nonzero. Furthermore $exy \leq ex$, so

$$0 \neq exyx^{-1} \leq exx^{-1} \leq e.$$

By minimality of e, it follows that $e = exyx^{-1}$. Therefore

$$ex = exyx^{-1}x = exx^{-1}xy = exy.$$

□

Proposition 3.2. *Let $(\mathcal{P}, S, \mathcal{E})$ be a patch system and $\leftrightarrow$ a commutation relation on $\mathcal{P}$. Let e be a minimal idempotent, and assume that $(\mathbf{p}, \mathbf{r}) \leftrightarrow^* (\mathbf{q}, \mathbf{s})$ where $\mathbf{c}$, $\mathbf{r}$, $\mathbf{q}$ and $\mathbf{s}$ are all in $\mathcal{P}^*$. If $\mathbf{cp}$ and $\mathbf{cq}$ are both e-sensible, then so are $\mathbf{cpr}$ and $\mathbf{cqs}$.*

Proof. Let $c = \mathcal{E}^*(\mathbf{c})$, $p = \mathcal{E}^*(\mathbf{p})$, and $q = \mathcal{E}^*(\mathbf{q})$. We have $ecpr = ecqs$ since $\leftrightarrow^*$ is effect-preserving, so it suffices to show that $ecpr$ is nonzero.

Since ecp is nonzero, by Proposition 1.3 we have $0 \neq ecpp^{-1} \leq ec$. But ec is minimal by Lemma 3.1; so $ecpp^{-1} = ec$. Then

$$0 \neq ecq = ecpp^{-1}q = ecprs^{-1}$$

since $p^{-1}q = rs^{-1}$ by the rotating and effect-preserving properties. Thus $ecpr$ is also nonzero. □

4. Permutivity

The previous section described how to merge changes from two distributed repositories. When three or more repositories are involved, we can apply the same technique by repeatedly performing pairwise merges until all of the repositories have been synchronized with each other.

However, although effect-preservation ensures that two merged histories must have the same effect, we cannot conclude the same when several repositories are merged together. In particular, suppose that three remote repositories each have distinct patches p_1, p_2 and p_3. Ideally, the order in which we merge those patches into our own repository should not change their overall effect. However, it is not possible to prove, e.g., that the order p_1, p_2, p_3 will have the same effect as p_2, p_1, p_3, if we only use the definitions from the previous sections. (A simple counter-example in a related setting is described by [6].)

It turns out that one last condition is required, known in the Darcs literature as *permutivity*. Informally, it states that if a patch sequence is rearranged by repeatedly commuting adjacent elements, then the resulting sequence should depend only on the final order of elements and not the choice of intermediate swaps.

Our formalization of permutivity is similar to [5], but highlights some additional mathematical structure.

Definition 4.1. Let G_n denote the group of permutations of sequences of n elements. Let T_n denote the set of all adjacent transpositions in G_n, i.e. those whose only effect is to swap two consecutive elements.

Definition 4.2. Let $\mathcal{P}$ be a patch set; then let $\mathcal{P}^n$ denote the set of sequences in $\mathcal{P}^*$ with length n. Let $\mathcal{S}(\mathcal{P}^n)$ be the inverse semigroup whose elements are partial injective functions on $\mathcal{P}^n$.

Definition 4.3. Fix a uniquely commutating relation $\leftrightarrow$. Define the associated $\mathcal{F}_n : T_n \to \mathcal{S}(\mathcal{P}^n)$ as follows. Fix $t \in T_n$ which swaps elements i and $i+1$. Then $\mathcal{F}_n(t)$ is the element of $\mathcal{S}(\mathcal{P}^n)$ which acts on $(p_1, \ldots, p_n)$ by commuting p_i and p_{i+1} when $\leftrightarrow$ permits it, and which is undefined otherwise.

We have $\mathcal{F}_n$ well-defined since $\leftrightarrow$ is uniquely commuting. Furthermore, the symmetric property implies that each $\mathcal{F}_n(t)$ is injective, and that $\mathcal{F}_n(t)^{-1} = \mathcal{F}_n(t^{-1})$.

Definition 4.4. We say that a uniquely commuting relation is $n-$*permutative* if

$$s_1 \cdots s_k = t_1 \cdots t_m \implies \mathcal{F}_n(s_1) \cdots \mathcal{F}_n(s_k) = \mathcal{F}_n(t_1) \cdots \mathcal{F}_n(t_m),$$

for any $s_i, t_j \in T_n$. (Note that the left-hand side is a composition of permutations, whereas the right-hand side is a composition of partial functions in $\mathcal{S}(\mathcal{P}^n)$.) We say that $\leftrightarrow$ is *permutative* if it is *n-permutative* for all $n > 0$.

Algebraically, permutivity means that $\mathcal{F}_n$ can be extended to a semigroup homomorphism from G_n to $\mathcal{S}(\mathcal{P}^n)$.

Permutivity may seem like a difficult property to verify in practice. Luckily, the following result, which was proved in [5], saves a great deal of work:

Theorem 4.1. *If a uniquely commuting relation is 3-permutative, then it is permutative.*

Checking permutivity only for sequences of length 3 is a much more tractible problem, if still somewhat tedious. It can be aided by an automated theorem prover, for example as in [6].

In future work, we hope to use the permutivity property to fully specify the n-way merge algorithm and show that it results in distributed histories with identical effects.

5. Managing conflicts

5.1. **Conflictors in Darcs.** The previous discussion of history merging is not directly applicable when patches in different repositories conflict — for example, if they make different changes to the same line of the same file. Specifically, the merge algorithm breaks down when we try to merge two patches p and q such that $\left(p^{-1}, q\right)$ does not commute.

Darcs solves that issue by add a new patch type, called a *conflictor* [19], which "forces" a merge between any two patches that would otherwise conflict. When

Darcs creates a conflictor, it undoes the effects of both conflicting patches and marks the file as needing resolution. The user is then expected to record a new patch which manually merges the conflicting changes.

We will denote a conflictor as a pair of patches $C(p, q)$. In practice, the representation that Darcs uses is more complicated. However, that simplification makes it simpler to demonstrate the usefulness of our inverse semigroup model. We will elaborate on avenues of future work in Section 5.4.

The best way to understand the semantics of conflictors is to consider how they are used in merges. Suppose that Alice has patch p and Bob has conflicting patch q. Recall that merging causes new patches to be added to the end of one's repository; thus Alice ends up with history $pC(p, q)$, while Bob has $qC(q, p)$.

As in Section 3, we guide ourselves by requiring that

$$(p, C(p, q)) \leftrightarrow (q, C(q, p)) \tag{2}$$

when p and q conflict. The hope is that by combining the other desired commutation properties with (2), we can derive the full semantics of conflictor commutation.

In particular, substituting p^{-1} for p and applying the rotating and symmetric properties produces

$$(p, q) \leftrightarrow \left(C(p^{-1}, q), C(q, p^{-1})^{-1}\right)$$

when (p, q) cannot be otherwise commuted. In other words, conflictors should extend commutation of patches to another relation which can commute any pair of elements.

5.2. **Conflictor effects.** In order to model conflictors using our semigroup framework, we must precisely describe their image under the effect map. In what follows, assume that we have a patch system with effect map $\mathcal{E}$. For brevity, we will use the notation $\hat{p} = \mathcal{E}(p)$.

We propose to take

$$\mathcal{E}(C(p, q)) = \hat{p}^{-1}\hat{q}\hat{q}^{-1}.$$

That effect makes the same change to the repository state as $\hat{p}^{-1}$, but restricts its range to be compatible with the domain of $\hat{q}$.

For example, if $\mathbf{c}$ is some previous history, we find that

$$\mathcal{E}^*(\mathbf{c}pC(p, q)) = \mathcal{E}^*(\mathbf{c})\hat{p}\hat{p}^{-1}\hat{q}\hat{q}^{-1}.$$

In terms of partial functions, that effect restricts the range of $\mathcal{E}^*(\mathbf{c})$ to be contained in $\mathrm{Dom}(p) \cap \mathrm{Dom}(q)$. In other words, it restricts the repository history such that either p or q could potentially be applied at the end.

It is straightforward to show that merger commutation is effect-preserving, since idempotents commute:

$$\mathcal{E}^*(pC(p, q)) = \hat{p}\hat{p}^{-1}\hat{q}\hat{q}^{-1} = \hat{q}\hat{q}^{-1}\hat{p}\hat{p}^{-1} = \mathcal{E}^*(qC(q, p)).$$

Additionally, there is an analogue of Proposition 3.2: if $\mathbf{c}p$ and $\mathbf{c}q$ are e-sensible for some minimal element e, then so are $\mathbf{c}pC(p, q)$ and $\mathbf{c}qC(q, p)$. As the proof is essentially the same, we will omit it for brevity.

5.3. **Inverse conflictors.** If conflictors are to be included in a patch system, then we ought to be able to take their inverse. The effect of such an inverse must be:

$$\mathcal{E}(C(p,q)^{-1}) = \mathcal{E}(C(p,q))^{-1} = (\hat{p}^{-1}\hat{q}\hat{q}^{-1})^{-1} = \hat{q}\hat{q}^{-1}\hat{p}.$$

This indicates that the inverse of a conflictor patch should be an additional type of object separate from conflictors themselves.

Happily, our model is consistent with the rotating property. Specifically, after applying that property to (2):

$$\left(q^{-1}, p\right) \leftrightarrow \left(C(q,p), C(p,q)^{-1}\right),$$

we can confirm that the commutation remains effect-preserving:

$$\begin{aligned}\mathcal{E}^*(C(q,p)C(p,q)^{-1}) &= \hat{q}^{-1}(\hat{p}\hat{p}^{-1})(\hat{q}\hat{q}^{-1})\hat{p} \\ &= \hat{q}^{-1}(\hat{q}\hat{q}^{-1})(\hat{p}\hat{p}^{-1})\hat{p} = \hat{q}^{-1}\hat{p}.\end{aligned}$$

5.4. **Issues with conflictors.** The main problem with the above approach is that conflictors can themselves conflict with other patches or with each other. For example, if p, q and r mutually conflict then attempting to merge r with $C(p,q)$ produces the conflictor $C(r, C(p,q))$. As a result, Darcs allows arbitrary nesting of conflictors within other conflictors.

Unfortunately, the algorithms used by Darcs to commute and merge nested conflictors are known to have exponential running time in some cases. In the past, that issue has significantly affected the reliability and performance of Darcs for certain large repositories. To the user, the software appears to unpredictably hang while synchronizing with another repository. As a result, some projects take care to avoid conflicts between repositories and instead merge conflicting changes by hand [12].

Over the last several years, work has been ongoing to develop a more sophisticated type of conflictor without the above problems. Darcs version 2.0, released in April 2008, contains a conflictor type which avoids nested conflicts in many common cases. Unfortunately, it is still possible to construct pathological cases requiring exponential time in the number of conflicting patches.

Camp [1, 11] is a version control system based on Darcs which aims to develop more robust and efficient algorithms. The advances of Camp are intended to eventually be integrated into Darcs itself. In particular, Camp contains a conflictor type which is believed to avoid nesting altogether. Ignoring the details, we denote those conflictors as

$$C(\mathbf{r}, X, \mathbf{p}, q).$$

Here q is the patch represented by this conflictor; $\mathbf{p}$ is the "context" of patches which q depends on but which are represented by other conflictors. The actual changes made to the state by the conflictor are $\mathbf{r}$; and X is a set of all previous patches that q conflicts with, plus their contexts.

We claim that our above model of conflictor effects can be extended to Camp by:

$$\mathcal{E}(C(\mathbf{r}, \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}, \mathbf{p}, q)) = \mathcal{E}^*(\mathbf{r}\mathbf{x}_1{\mathbf{x}_1}^{-1} \cdots \mathbf{x}_n{\mathbf{x}_n}^{-1}\mathbf{p}qq^{-1}\mathbf{p}^{-1}).$$

We hope to further explore this approach in a future work, merging Camp's existing theoretical framework with our own to produce a more robust and comprehensible theory of patches.

6. Related work and conclusions

6.1. **Previous Darcs formalizations.** Our model of patches and patch sequences is most closely related to [11]. That paper defines concepts such as inverses and sensibility using axioms directly related to the algorithms of Darcs. For example, our Corollary 1.4 is taken as part of the definition of sensibility. In contrast, we have shown how that result follows from patch effects being injective (and thus forming an inverse semigroup).

Previous work [2, 18] has formalized patches using a different approach. Patches are represented by a change (for example, "add file F") plus an input and output context. The contexts essentially model the complete state of the repository (including files other than F) before and after the patch was applied. The usual notation for a patch under that formalism is ${}^{o}A^{a}$, where A is the change and o, a are the input/output contexts.

Patches in those works can be only be composed if the contexts match up; we may compose ${}^{o}A^{a}$ and ${}^{a'}B^{b}$ into ${}^{o}AB^{b}$ only when $a = a'$. Contrast that formalism with [11] and our own, in which any two patches may be composed but only certain sequences are labelled "sensible." The inverse of a patch ${}^{o}B^{b}$ is a patch ${}^{b}\overline{B}^{o}$ with swapped contexts whose change $\overline{B}$ undoes the change made by B. The definitions of commutation, permutation, etc. similarly require that contexts match up correctly, so that all operations will make consistent changes to the repository state.

Löh et. al. [10] model patches as a triple of sets. As in the previous work on Darcs patch theory, only certain compatible pairs of patches may be composed. They note that their definitions form a *groupoid*, i.e. a category in which each morphism has an inverse. The previous Darcs patch theory can also be adapted to the groupoid setting; the patch contexts are category-theoretic *objects*, and the patches are *morphisms* between the contexts.

In fact, that approach turns out to be dual to our own. The Ehresmann-Schein-Nambooripad theorem [8] states that inverse semigroups are equivalent to *inductive groupoids* (essentially, groupoids with an ordering similar to natural ordering we've previously discussed). For example: if X is the semigroup of partial one-to-one functions on a set S, then objects in the associated groupoid are subsets of X; the morphisms between two subsets A and B are partial functions f such that $\mathrm{Dom}(f) = A$ and $\mathrm{Ran}(f) = B$; and fg is defined when $\mathrm{Ran}(f) = \mathrm{Dom}(g)$.

That correspondence suggests that the separate patch theories are merely different ways of viewing the same underlying mathematical structure. We find our model more useful for some theoretical analyses, since it allows us to model patches and sensibility as separate mathematical concepts. In contrast, [2] embeds patch contexts in the type system. That approach enables the type-checker to confirm at compile-time that the commutation and merging functions only produce sensible compositions.

6.2. **Alternate systems.** The core idea of Darcs, that changes from concurrent systems must be transformed in order to be meaningfully combined, has appeared in several other contexts. Prakash and Knister [15] defined a *Transpose* function analogous to Darcs' commutation. They used that function to implement a "selective undo" feature, which can undo an operation in the middle of a history of commands. As in Darcs, they enforce dependencies between operations by only allowing certain pairs of operations to be transposed.

Roscoe [17] presented an algorithm for synchronizing a certain arrangement of distributed databases. He modelled database updates abstractly as elements of an algebraic group. In particular, he used the mathematical concept of group *conjugation* to describe abstractly how operations can be transformed to become compatible with concurrent operations. That theory does not directly apply to Darcs patches, which do not have inverses in the sense of groups. However, it would be interesting to study further whether some analogue of conjugation can give better insight into the properties of Darcs commutation.

Ellis and Gibbs [4] defined *operational transformations* for use in collaborative editors which let users view each others' changes in real-time. A *transformation function* $T(o_2, o_1)$ takes two concurrent operations o_1, o_2 and produces a new operation o_2' which may be performed after o_1 while maintaining the original intent. Ressel et. al. [16] proposed several properties which T ought to satisfy. In particular, they showed that correctness of their synchronization algorithm follows from the following claims:

TP1: $o_1 T(o_2, o_1) \equiv o_2 T(o_1, o_2)$.
TP2: $T(o_3, o_1 T(o_2, o_1)) = T(o_3, o_2 T(o_1, o_2))$.

Note that their TP1 is similar to the commutation $(p, r) \leftrightarrow (q, s)$ which we used in Section 3 to merge two patches p and q. Additionally, TP2 is similar to our 3-permutivity (Section 4). The other properties listed by [16] also correspond to those of Darcs's patch theory (symmetry, rotating, etc.), but are based upon the merging operation T instead of the commutation operation $\leftrightarrow$.

It turns out that finding transformation functions provably satisfying TP2 is difficult in practice. Imine et. al. [6] achieved TP2 by keeping track of deleted characters in the distributed state. Methods have also been developed which do not require TP2 to achieve consistent synchronization; [13] uses a central server, while [23] assigns a global timestamp to each operation.

Sun et. al. [22, 21] have noted that even if TP1 and TP2 are satisfied, the final shared state may not preserve the meaning of the original operations. They proposed the CCI framework which adds an *intention preservation* requirement. Li and Li [9] have formalized that property within the CSM framework, which requires that the transformation function preserve a total ordering among operations, called an *effects relation*. For example, when the shared state is an array of characters, the effects relation orders changes according to the position of the character that they affect.

In a future work, we plan to model operational transformations using the same semigroup-based methods that we have applied to Darcs. We suggest that using a generic, established mathematical model such as semigroup theory may lead to a better understanding of the connections between the different formalisms which have been developed so far.

6.3. **Conclusion.** In this paper, we have described how Darcs uses a theory of patches to handle the different tasks of revision control. We have connected that theory to established mathematical objects by modelling patch effects using semigroup theory. In particular, we have presented novel formulations of the concepts of patch inversion and sequence sensibility.

We have shown that our model leads to clean formal descriptions of the commutation and merging operations which form the basis for the algorithms of Darcs. Additionally, we have proven the following results within our model:

- It is always sensible to apply the inverse of the patches at the end of a repository's history (Corollary 1.4).
- A commutation relation on patches can be naturally extended to a relation on patch sequences with the same properties (Proposition 2.1).
- Merging patches from a remote user produces a sensible patch sequence (Proposition 3.2).

Finally, we have used our theory to gain new insight into two important parts of patch theory. We have given a concise statement of the permutivity property, which helps derive the sensibility of merges between more than two parties. We have also given new justification for and understanding of conflictor patches, which are used to resolve conflicting changes from different sources. We hope that our work will help strengthen the theoretical foundations of a future version of Darcs.

Acknowledgements

The author was supported in part by NSF Grant Number DMS-0502315. Clint Givens gave helpful suggestions on the presentation of this paper.

References

[1] Camp home page. http://projects.haskell.org/camp.
[2] Jason Dagit. Type-correct changes — a safe approach to version control implementation. Master's thesis, Oregon State University, March 2009.
[3] Darcs home page. http://darcs.net.
[4] C Ellis and S Gibbs. Concurrency control in groupware systems. *SIGMOD '89: Proceedings of the 1989 ACM SIGMOD international conference on Management of data*, Jun 1989.
[5] Ganesh Sittampalam et al. Some properties of darcs patch theory. Available from http://urchin.earth.li/darcs/ganesh/darcs-patch-theory/theory/formal.pdf, November 2005.
[6] Abdessamad Imine, Michal Rusinowitch, Grald Oster, and Pascal Molli. Formal design and verification of operational transformation algorithms for copies convergence. *Theoretical Computer Science*, 351(2):167–183, 2006.
[7] Inverse semigroups. Available from http://en.wikipedia.org/wiki/Inverse_semigroup, April 2009.
[8] Mark V. Lawson. *Inverse Semigroups*. World Scientific, New Jersey, 1998.
[9] Rui Li and Du Li. A new operational transformation framework for real-time group editors. *Parallel and Distributed Systems, IEEE Transactions on*, 18(3):307–319, 2007.
[10] Andreas Löh, Wouter Swierstra, and Daan Leijen. A principled approach to version control. In FASE, 2007. Available from http://www.cs.nott.ac.uk/~wss/Publications/fase07.pdf.
[11] Ian Lynagh. Camp patch theory. Available from http://projects.haskell.org/camp/files/theory.pdf, April 2009.
[12] Simon Marlow. Guidelines for using darcs with ghc. Available from http://hackage.haskell.org/trac/ghc/wiki/WorkingConventions/Darcs#Conflicts, 2009.
[13] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 111–120, New York, NY, USA, 1995. ACM.
[14] Perforce home page. http://www.perforce.com/.
[15] Atul Prakash and Michael J. Knister. Undoing actions in collaborative work: Framework and experience. Technical report, Department of Electrical Engineering and Computer Science, University of Michigan, 1994.
[16] Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. *CSCW '96:*

Proceedings of the 1996 ACM conference on Computer supported cooperative work, Nov 1996.

[17] A. W. Roscoe. Consistency in distributed databases: A group-like algebra and its applications. Technical Report PRG87, Oxford University Computing Laboratory, 1990.

[18] David Roundy. Implementing the darcs patch formalism. Available from http://web.archive.org/web/*/http://darcs.net/fosdem_talk/talk.pdf, 2006.

[19] David Roundy. Theory of patches. Available from http://darcs.net/manual/node9.html, April 2009.

[20] Subversion home page. http://subversion.tigris.org/.

[21] Chengzheng Sun and David Chen. Consistency maintenance in real-time collaborative graphics editing systems. *ACM Transactions on Computer-Human Interaction*, 9:1–41, 2002.

[22] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *Transactions on Computer-Human Interaction (TOCHI)*, 5(1), Mar 1998.

[23] Nicolas Vidot, Michelle Cart, Jean Ferrié, and Maher Suleiman. Copies convergence in a distributed real-time collaborative environment. *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work*, Dec 2000.

Department of Mathematics, UCLA, Los Angeles, CA 90095-1555
E-mail address: jjacobson@math.ucla.edu