An Algebra of Patches

Ian Lynagh

October 30, 2006

Abstract

In this paper we give an overview of how Darcs, a revision control system, works. In particular we focus on its "algebra of patches" which dictates its behaviour. We describe an unsolved problem, conflicting patches, and go into detail on an appealing avenue of investigation which has thus far not been successful.

1 Colour

This paper colours terms depending on what sort of thing they represent. While the colour is mostly redundant, it will make it much easier to follow if you view this document in colour.

2 Introduction

Most revision control systems, such as CVS, subversion and arch, are based on *file trees*. The current state of the repository is a file tree and when you commit a change that file tree is updated to the one you commit. By contrast, the Darcs revision control system is based on *patches*, i.e. the changes between the file trees corresponding to the history of file trees in the repository. As we will see, this makes some things easy which are either hard or impossible in other revision control systems. This approach is also easier to reason and prove properties about.

However, there is one problem that is unsolved in the Darcs world, and that is how to deal with conflicting patches. The existing algorithm is both incomplete and, in certain use cases, exponential. While many people are able to work with Darcs in a way that avoids, or works around, these problems, this remains a large issue in the Darcs community.

The remainder of this paper is split into three sections. In Section 3 we start from a high-level view of Darcs and proceed to zoom in on the underlying patch algebra, describing what it does and how it does it.

Then, in Section 4, we take a look at the problem with conflicting patches, what properties it needs to satisfy, and explain the form we would like the solution to take.

Finally, in Section 5 we describe the state of the art in what we have found along these lines. We give datastructures for describing conflicts that initially seem plausible, but upon digging deeper it turns out that there are cases that they are not able to handle.

3 What works

We begin by starting from a very high-level view of Darcs, and then working down towards the underlying algebra of patches.

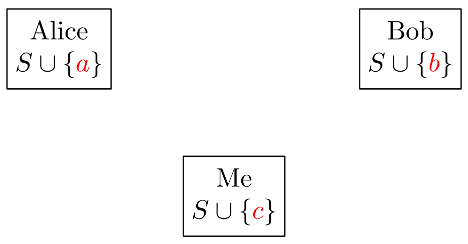
3.1 Repository networks

Darcs is a *distributed* revision control system, which is to say that there can be many repositories but there is no structure enforced between them. In particular, Darcs does not require that you have a single, central repository. Darcs is also *symmetric*, i.e. if you have two repositories then Darcs does not require that one be the “master” repository and the other be a “slave” repository. It is, of course, possible to use Darcs in a centralised manner, or to arrange one’s repositories in a hierarchial manner, if one so desires.

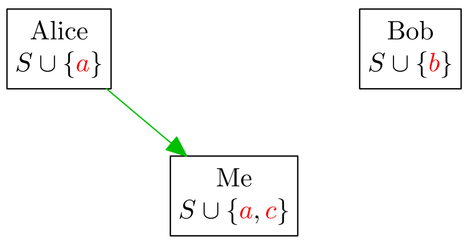
3.2 Sets of patches

The state of a repository in Darcs is defined by the set of patches that it contains. For example, consider the repositories in Figure 1a. Alice, Bob and myself all have all the patches in the set S. Additionally, Alice has recorded a patch called a (lowercase red letters are *patch names*; we will assume that all patches have a unique name for this paper), Bob has recorded a patch called b and I have recorded a patch called c. All of our repositories are thus in different states, as indicated in the figure.

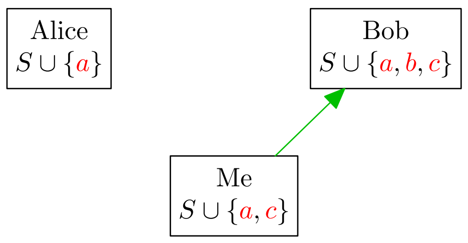
I can then *pull* the patch a from Alice, resulting in the situation shown in Figure 1b. Similarly I can *push* the patches a and c to Bob, illustrated in Figure 1c. Finally, if I pull from Bob then both Bob and I end up with the set of patches $S \cup \{a, b, c\}$ in our repositories as shown in Figure 1d. Therefore our repositories are in the same state. This means that any file must have



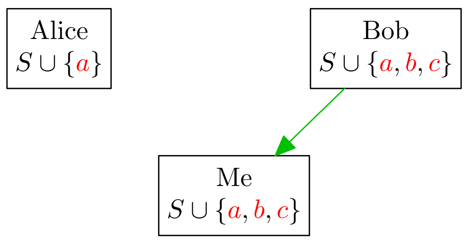
(a) Initial state



(b) Pulling from Alice



(c) Pushing to Bob



(d) Pulling from Bob

Figure 1: Pushing and pulling between repositories

the same contents in Bob's repository as it does in my repository. Note that this is the case despite the fact that, for example, I had c before getting b while Bob had b before getting c.

Darcs is *symmetric* in its handling of patches, i.e. suppose two people each have a repository in the same state. If they both record a patch in their repository then there is no ordering relation that says that one happened before the other, or that one takes precedence over the other. In particular, if the two patches conflict then we do not allow ourselves to look inside the conflicting patches to determine what action we should take; rather, our action should be entirely symmetric.

3.3 Sequences of patches and commutation

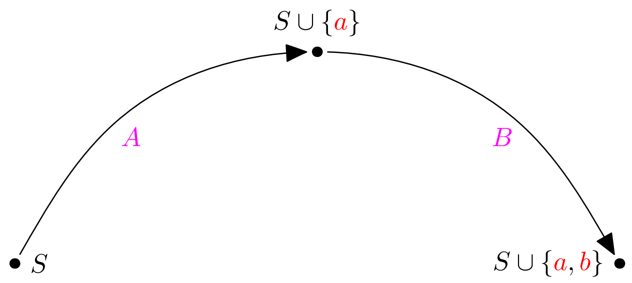
While thinking of sets of patches works well as a way of thinking about the differences between repositories, it is impractical to actually work with patches this way. Instead, we work with sequences of patches. This is illustrated abstractly in Figure 2a. We start, on the left, with a repository containing the set of patches S. Then we have the patch A (purple uppercase letters represent actual patches, with both a patch name and an *effect*) which takes us to the state containing everything in S as well as a (to simplify the notational overhead we use a for the patch name of the patch A, b for the patch name of the patch B, and so on). Finally the patch B takes us to the state $S \cup \{a, b\}$.

In Figure 2b we make this example concrete. We assume that the patches in S result in a file containing lines "X", "Y" and "Z", that A adds a line "A" as line 2 and that B adds a line "B" as line 4.

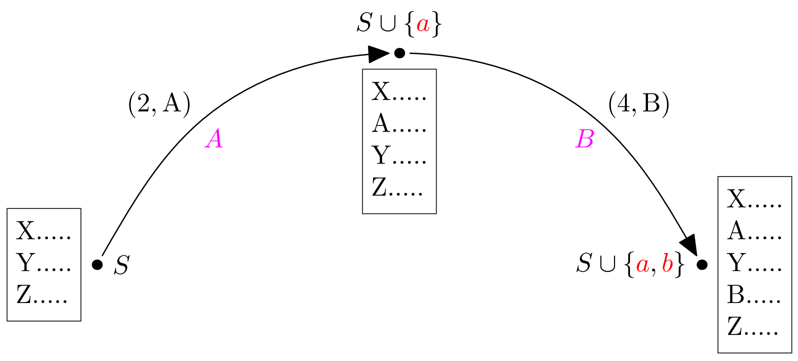
What is interesting is that we can *commute* the patches A and B, as shown in Figure 2c. Here we have created two new patches, B' and A' (if a patch is decorated with primes then its patch name is not altered, e.g. here B' also has patch name b and A' also has patch name a, but its effect may differ) which in some sense (which is hard to formally specify) are *the same* patches as B and A respectively. However, these new patches apply in different *contexts* (the terms "repository state" and "context" are interchangeable; sometimes one flows more naturally than the other). We say that we have *commuted* the *patch sequence* AB to get the patch sequence $B'A'$.

Note that the patch sequences AB and $B'A'$ have the same start and end context, but go via a different intermediate context. Note also that the internal representation and the externally visible *effect* of the patches may change, e.g. here B' adds "B" at line 3 rather than line 4.

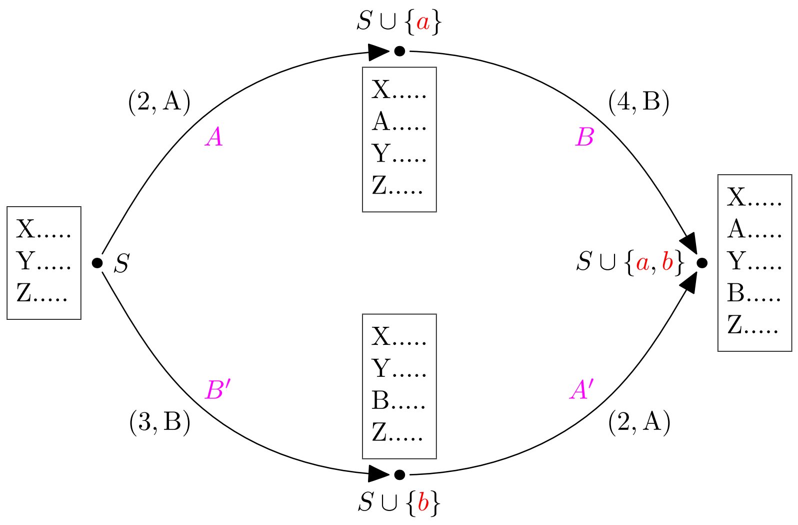
An abstracted diagram of patch commutation is shown in Figure 3; while this doesn't show anything new, we will want to refer back to it later in this



(a) The abstract sequence



(b) The concrete sequence



(c) Commuting the sequence

Figure 2: A concrete example of commutation

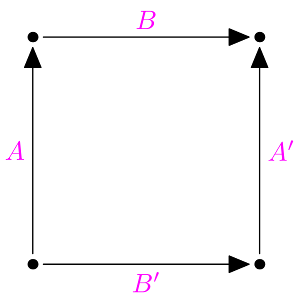


Figure 3: Patch commutation

paper.

We write $AB \leftrightarrow B'A'$ if the patch sequence AB commutes to give us $B'A'$. We also write $Xs \leftrightarrow Ys$ if the patch sequences Xs and Ys differ only in that an adjacent pair of patches have been commuted. We write $Xs \leftrightarrow^* Ys$ if it is possible to reach Ys from Xs by doing only an arbitrary number of commutes of adjacent patches.

We now have enough to start writing down some axioms and corollaries.

Axiom 3.1
$AB \leftrightarrow B'A' \iff B'A' \leftrightarrow AB$

Axiom 3.1 tells us that if we commute two patches then we can commute the result and get back to where we started (note that the $\iff$ symbol is the normal logical "if and only if", and unrelated to the $\leftrightarrow$ relation).

Axiom 3.2
All patches are invertible, and the inverse is both its left and right inverse when sequentially composed.

Axiom 3.2 tells us that given any patch A we can find its inverse, written A^{-1} (which has patch name a^{-1}). Both the sequence AA^{-1} and the sequence $A^{-1}A$ have no effect on the context (but, of course, they can only be applied to a particular context, just as any other patch or patch sequence).

Axiom 3.3
$AB \leftrightarrow B'A' \iff A^{-1}B' \leftrightarrow BA'^{-1}$

Corollary 3.4

$$\begin{aligned} & AB \leftrightarrow B'A' \\ \iff & BA'^{-1} \leftrightarrow A^{-1}B' \\ \iff & A'^{-1}B'^{-1} \leftrightarrow B^{-1}A^{-1} \\ \iff & B'^{-1}A \leftrightarrow A'B^{-1} \end{aligned}$$

$$\begin{tikzcd}
 & ABC \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
AC'''B''' \arrow[d, leftrightarrow] & & B'A'C \arrow[d, leftrightarrow] \\
C''A'''B''' \arrow[dr, leftrightarrow] & & B'C'A'' \arrow[dl, leftrightarrow] \\
 & C''B''A'' &
\end{tikzcd}$$

Figure 4: Circular commutations

Axiom 3.3 says that if we can commute as described by the diagram in Figure 3, then we can also start from the top-left corner rather than the bottom-left corner and do an equivalent commutation, using the patch inverse when we have to go against an arrow. Corollary 3.4 generalises this to say that we can start at any corner of the diagram.

Axiom 3.5

$$\begin{aligned}
&\begin{pmatrix} ABC \leftrightarrow B'A'C & \leftrightarrow B'C'A'' & \leftrightarrow C''B''A'' \\ \quad\;\;\, \leftrightarrow C''A'''B''' & \leftrightarrow A''''C'''B''' & \leftrightarrow A''''B''''C'''' \end{pmatrix} \\
\Rightarrow\ & A'''' = A \wedge B'''' = B \wedge C'''' = C
\end{aligned}$$

Axiom 3.5 is best illustrated by the diagram in Figure 4. The axiom says that if we start with a patch sequence ABC and we walk through all the possible permutations of those patches by doing pairwise commutes, then we end up back where we started.

Theorem 3.6
Axiom 3.5 generalises to any sequence of commutations of arbitrarily many patches that returns the patch names to their initial order.

3.4 Cherry picking

So far we have spent a lot of time talking about what commutation does and what properties it has, but we have not yet given any motivation for why it is useful. Consider this scenario:

- I have a repository containing a single patch: A
- You pulled from my repository and recorded two further patches, B and C: ABC.

- I don't like your patch B; perhaps it is an experimental feature, for example. However, your patch C is a fix for a critical bug, so I *do* want C.

This ability to take patches of your choice, regardless of the order they were created in, from other repositories is refered to as *cherry picking*.

At this point we introduce the notation $\{ab\}$ for contexts; this is equivalent to the $\{a, b\}$ notation used earlier, but slightly more space efficient and the colouring makes it easier to read expressions containing both contexts and patches.

We can now decorate the repositories in our cherry-picking problem with contexts. I have the repository $\{\}A\{a\}$ while you have $\{\}A\{a\}B\{ab\}C\{abc\}$, and I would like to create the repository $\{\}A\{a\}\{ab\}C\{abc\}\{ac\}$. As we can see, there are context mismatches, so just plugging the patches together like this is not permitted. The solution is to first perform the commutation

$$\{a\}B\{ab\}C\{abc\} \leftrightarrow \{a\}C'\{ac\}B'\{abc\}$$

and we can then discard B' and construct the repository $\{\}A\{a\}C'\{ac\}$.

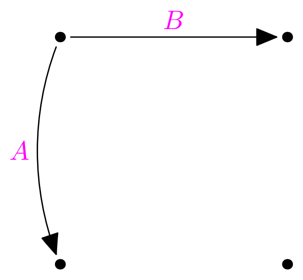
3.5 Merging

Cherry picking is one of two complex inter-repository operations that one might want to do. The other, which we shall describe now, is *merging*. Consider this scenario:

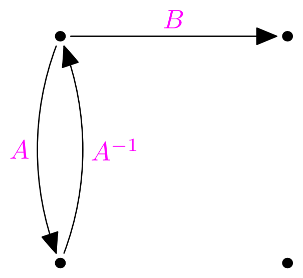
- I have a repository containing a single patch: A
- You also have a repository containing a single patch: B.
- I like your patch and want to pull it into my repository.

Again, just creating a patch sequence would produce a context mismatch: $\{\}A\{a\}\{\}B\{b\}$. To see how to proceed, let us consider the diagram in Figure 5a. Here we have the start state, in this case the empty context, in the top left-corner, and two patches which take us to two different states. Our aim is to get to the bottom-right corner, where we have applied both patches, but currently we don't know how to get there.

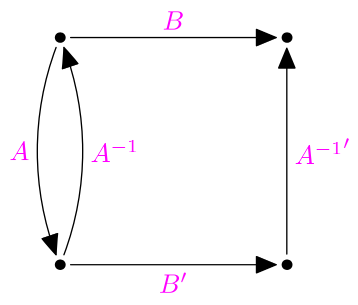
However, by filling in the inverse of one of the patches, as shown in Figure 5b, we see that we can actually consider it instead as the patch sequence $A^{-1}B$. We can then fill in the rest of the commutation diagram for this sequence, as shown in Figure 5c, from which we find that if we do the commute $A^{-1}B \leftrightarrow B'A^{-1'}$ then AB' has the correct contexts.



(a) A merge



(b) Adding one of the inverses



(c) Filling in the commutation square

Figure 5: Merging by commutation

Therefore, using $+$ for the merge operator, we define $A + B = AB'$ where $A^{-1}B \leftrightarrow B'A^{-1'}$.

Another way of thinking about this is to first construct the valid sequence $AA^{-1}B$, then to commute A^{-1} and B in this sequence $AA^{-1}B \leftrightarrow AB'A^{-1'}$ before finally discarding the $A^{-1'}$ patch leaving us with AB' as before.

3.6 Summary

We have described an algebra of patches, centered around a comcept of commutation. We have defined a set of axioms that the primitive patches and this commutation operator must obey.

Everything up to this point follows naturally from the initial design decisions to build a symmetric, distributed, patch-based revision control system.

4 What we can't do

So far everything seems fine but, as hinted at earlier, there is a problem. We have assumed that if we have two patches in sequence then we will always be able to commute them, but that is not always the case, and therein lies our troubles.

Before discussing this, we first introduce two new bits of notation. We use blue upper case letters to represent the effect of a patch, and just as we use a for the name of A, we use A for the effect of A. Note that, while A' shares the name a, it has its own effect A'.

We can thus think of a patch A as being composed of two parts, its name a and its effect A.

We also use an uppercase red letter to mean a patch with the appropriate name, without saying anything about the effect of the patch. For example, A has the name a but does not in general have the effect A.

4.1 Dependencies

First, let's consider a cherry picking example. Suppose we have this scenario:

- You have two patches: $(a, \text{create file f})(b, \text{put content in f})$
- I only want the second patch: B

As you would expect, it is not possible to commute these two patches, as it doesn't make sense to put content in a file before that file is created. But that does not cause a problem for Darcs: it is entirely reasonable to refuse to pull one patch, but not another that the first *depends* on.

4.2 Conflicts

Now let's see what happens in a merging scenario:

- I have $(a, \text{put X in file f})$
- You have $(b, \text{put Y in file f})$
- I want to have both patches: AB

But I can't have both! If we try to compute $A + B$ then the underlying commute will fail as a *conflicts with* b. Unfortunately, in this case it is *not* OK; it is unreasonable to require a be removed when we pull an additional patch b.

4.3 Properties of Dependencies and Conflicts

As you would expect from the definition of merging in terms of commutation, the concepts of dependency and conflict are related. To be precise:

$$\begin{aligned} & a \text{ conflicts with } b \\ \iff & b \text{ conflicts with } a \\ \iff & b \text{ depends on } a^{-1} \\ \iff & a \text{ depends on } b^{-1} \end{aligned}$$

To get some intuition for why this is so, the diagram in Figure 5c visualises why a conflict between a and b is the same thing as a dependency of b on a^{-1}.

Axiom 4.1
If $AB \leftrightarrow B'A'$ and $ABC \leftrightarrow^{*} C'A''B''$ then $A''B'' \leftrightarrow B'''A'''$.

Recall our circular sequence of commutations from Figure 4. Axiom 4.1 says that if half of the commutation arrows exist then all of them must exist. For example, if the blue arrows in Figure 6 exist then a and b commute, and we can commute c through them. Therefore they should still commute after we have done so, so the red arrow also exists. By applying the same logic twice more we can show that all the arrows exist.

4.4 Design Choices

Let us return to our conflicting merge scenario:

- I have A

$$\begin{tikzcd}
 & ABC \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
AC'''B''' \arrow[d, leftrightarrow] & & B'A'C \arrow[d, leftrightarrow] \\
C''A'''B''' \arrow[dr, leftrightarrow] & & B'C'A'' \arrow[dl, leftrightarrow] \\
 & C''B''A'' &
\end{tikzcd}$$

Figure 6: Partial circular commutations

- You have B
- I want to keep my patch A, but also pull your patch B

Here we make a design choice: The result of doing this merge will be the patch A followed by a *conflictor* with name b. A conflictor is a special type of patch we are introducing to help us represent the result of a conflicted merge. Thus the result of $A + B$ is AB regardless of whether a and b conflict or not.

But what should B be in the conflicting case? What should it do?

Let us answer the second question first. We will write $\mathcal{E}(Ps)$ for the *effect* of the patch sequence Ps. So what should $\mathcal{E}(AB)$ be? There are only three possibilities which seem plausible that are available to us: $\mathcal{E}(AB) = A$, $\mathcal{E}(AB) = B$ and $\mathcal{E}(AB) = \epsilon$, with ϵ meaning no effect. But we require $A + B = B + A$ and our requirement that we treat patches symmetrically means that $\mathcal{E}(AB) = \epsilon$ is the only consistent solution. But we know that $\mathcal{E}(A) = A$, so we can deduce that $\mathcal{E}(B) = A^{-1}$.

We now introduce the notation $[\![n; Es]\!]$ to mean "a conflictor with name n and effect Es". Note that it is not necessarily the case that two such patches which are written the same are identical.

Then we can say $A + B = A\,[\![b; A^{-1}]\!]$ and, by translating back to our definition of merging in terms of commutation, we get $A^{-1}B \leftrightarrow [\![b; A^{-1}]\!]\,[\![a^{-1}; B]\!]$. This is a bit hard to think about due to the inverses, so we can substitute in different patches to get $XY \leftrightarrow [\![y; X]\!]\,[\![x; Y]\!]$.

4.5 Larger Conflict Cases

We extend our decision that the merge of two conflicting patches has no effect to a design principle: the effects of any patches that are in conflict with any other patches in our repository are not applied.

Now let us consider a larger case: suppose we want to compute $AB + C$, where C conflicts with A and B depends on A. This gives us $AB+C = ABC$.

Our design principle tells us that the total effect of the result should be ϵ, so we have $\mathcal{E}(ABC) = \epsilon$. Therefore we get $AB + C = AB [\![c; B^{-1}A^{-1}]\!]$.

Going back to the underlying commutations, we already know from the previous case that $B^{-1}A^{-1}C \leftrightarrow B^{-1} [\![c; A^{-1}]\!] [\![a^{-1}; C]\!]$, and now we can deduce that $B^{-1} [\![c; A^{-1}]\!] [\![a^{-1}; C]\!] \leftrightarrow [\![c; B^{-1}A^{-1}]\!] [\![b^{-1}; \epsilon]\!] [\![a^{-1}; C]\!]$. Again renaming to eliminate inverses we get $XYZ \leftrightarrow X [\![z; Y]\!] [\![y; Z]\!] \leftrightarrow [\![z; XY]\!] [\![x; \epsilon]\!] [\![y; Z]\!]$.

4.6 Symmetry by Inversion

We can get a symmetrical result to the last one by inverting all the sequences, expanding and renaming; indeed, this technique is applicable to any such derivation we do. In this case it goes like this:

Start with the law we have developed:
$XYZ \leftrightarrow [\![z; XY]\!] [\![x; \epsilon]\!] [\![y; Z]\!]$

Invert both sides:
$(XYZ)^{-1} \leftrightarrow ([\![z; XY]\!] [\![x; \epsilon]\!] [\![y; Z]\!])^{-1}$

Expand all the inverses:
$Z^{-1}Y^{-1}X^{-1} \leftrightarrow [\![y^{-1}; Z^{-1}]\!] [\![x^{-1}; \epsilon]\!] [\![z^{-1}; Y^{-1}X^{-1}]\!]$

Rename to eliminate inverses:
$PQR \leftrightarrow [\![q; P]\!] [\![r; \epsilon]\!] [\![p; QR]\!]$

4.7 Putting it all together

By combining this new law and its symmetric sibling, we get

$$ABC \leftrightarrow^* [\![c; AB]\!] [\![b; B^{-1}]\!] [\![a; BC]\!]$$

We already knew the result of commuting a patch left past two patches it depends on, and the result of commuting a patch right past two patches that depend on it. As we also know the contexts for the middle patch, it is easy to work out what its effect must be.

5 An idea that doesn't work

We will now look at the examples we have created to get some intuition for how the commutation algorithm must work.

First consider

$$ABC \leftrightarrow A [\![c; B]\!] [\![b; C]\!] \leftrightarrow [\![c; AB]\!] [\![a; \epsilon]\!] [\![b; C]\!]$$

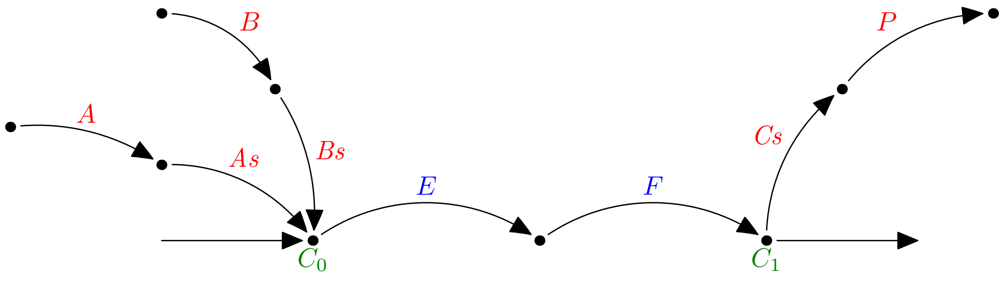


Figure 7: A left conflictor

As the patch c commutes left past patches it depends upon, it gathers up their effects. Symmetrically, when b commutes right past a patch that depends upon it, it gathers up its effect. When a commutes right past the same patch it doesn't gather the effect as another patch has already done so.

Now consider

$$ABC \leftrightarrow [\![c; AB]\!]\ [\![b; B^{-1}]\!]\ [\![a; BC]\!]$$

Again a and c have gathered the effects of the patches they pass as before, but here the patch b has had its effect gathered both by a patch moving left and a patch moving right. Therefore its effect is being applied twice, once to its left and once to its right. Thus it needs to unapply its effect so that the effect is applied once in total.

With that in mind, we will have three types of conflictor. Two of these, *left* and *right* conflictors, are symmetric; for example, if b depends on a then we say $AB \leftrightarrow [\lhd A \blacktriangleleft \prec B][A \succ \blacktriangleright B \rhd]$, where $[\lhd A \blacktriangleleft \prec B]$ is the left conflictor implementing $[\![b; A]\!]$ and $[A \succ \blacktriangleright B \rhd]$ is the right conflictor implementing $[\![a; B]\!]$. In general, left conflictors look like

$$[A \succ As, B \succ Bs \lhd EF \blacktriangleleft Cs \prec P]$$

This patch has the name p, conflicts with and applies the effects of e and f (or any number of patches in general), and conflicts with but does not apply the effects of a and b (i.e. those effects are applied by another left conflictor further to its left) (and again, there can be any number of patches in general). The patch sequences are only there to make the contexts work out. Here is the same left conflictor with the contexts annotated:

$$C_0[A \succ AsC_0, B \succ BsC_0 \lhd C_0EFC_1 \blacktriangleleft C_1Cs \prec P]C_1$$

but it is probably more useful to instead look at the graphical version in Figure 7. Right conflictors are simply the symmetric counterparts of left conflictors.

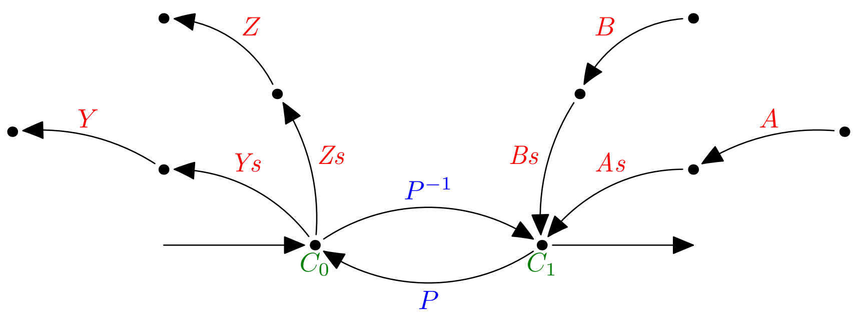


Figure 8: A middle conflictor

The final type of conflictor is a *middle* conflictor.

$$[A \succ As, B \succ Bs, \lhd P \rhd Ys \prec Y, Zs \prec Z]$$

This is a patch with name p, which has conflicted with a and b as it was going left, and conflicted y and z as it was going right. Again, there can be any number of patches on either side in general. The patch sequences are again just for satisfying contexts. The effect of this middle conflictor is P^{-1}. Here it is again with context annotations:

$$C_0[A \succ AsC_1, B \succ BsC_1 \lhd C_1PC_0 \rhd C_0Ys \prec Y, C_0Zs \prec Z]C_1$$

but again, the graphical version in Figure 8 is probably easier to understand.

5.1 An Example

As an example of the conflictors, here is a sequence of commutes for reversing the sequence ABC, where each patch depends on those to its left:

$$\begin{aligned}
&\{\}A\{a\}B\{ab\}C\{abc\}\\
\leftrightarrow\ &\{\}[\lhd A \blacktriangleleft \prec B]\{a\}[A \succ \blacktriangleright B \rhd]\{ab\}C\{abc\}\\
\leftrightarrow\ &\{\}[\lhd A \blacktriangleleft \prec B]\{a\}[A \succ \lhd \blacktriangleleft B \prec C]\{a\}[A \succ \blacktriangleright BC \rhd]\{abc\}\\
\leftrightarrow\ &\{\}[\lhd AB \blacktriangleleft \prec C]\{ab\}[A \succ \lhd B \rhd \prec C]\{a\}[A \succ \blacktriangleright BC \rhd]\{abc\}
\end{aligned}$$

5.2 The problem

Finally, we come to a demonstration of the problem. Figure 9 shows all the permutation of the patch sequence $ABCD$, where each patch depends on those to its left.

The problem is that there are four cases in which we are unable to fill in the answer, written $***p***$ in the figure. These are the four cases where

A	$abcd = \{\}$	A	$\{a\}$	B	$\{ab\}$	C	$\{abc\}$	D	$\{abcd\}$
1	$abdc = \{\}$	A	$\{a\}$	B	$\{ab\}$	$[\lhd C \blacktriangleleft \prec D]$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
B	$acbd = \{\}$	A	$\{a\}$	$[\lhd B \blacktriangleleft \prec C]$	$\{ab\}$	$[B \succ \blacktriangleright C \rhd]$	$\{abc\}$	D	$\{abcd\}$
2	$acdb = \{\}$	A	$\{a\}$	$[\lhd B \blacktriangleleft \prec C]$	$\{ab\}$	$[B \succ \lhd \blacktriangleleft C \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD \rhd]$	$\{abcd\}$
3	$adbc = \{\}$	A	$\{a\}$	$[\lhd BC \blacktriangleleft \prec D]$	$\{abc\}$	$[B \succ C \blacktriangleright \rhd \prec D]$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
4	$adcb = \{\}$	A	$\{a\}$	$[\lhd BC \blacktriangleleft \prec D]$	$\{abc\}$	$[B \succ \lhd C \rhd \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD \rhd]$	$\{abcd\}$
1	$bacd = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$[A \succ \blacktriangleright B \rhd]$	$\{ab\}$	C	$\{abc\}$	D	$\{abcd\}$
C	$badc = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$[A \succ \blacktriangleright B \rhd]$	$\{ab\}$	$[\lhd C \blacktriangleleft \prec D]$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
3	$bcad = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$[A \succ \lhd \blacktriangleleft B \prec C]$	$\{a\}$	$[A \succ \blacktriangleright BC \rhd]$	$\{abc\}$	D	$\{abcd\}$
4	$bcda = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$[A \succ \lhd \blacktriangleleft B \prec C]$	$\{a\}$	$[A \succ \lhd \blacktriangleleft BC \prec D]$	$\{a\}$	$[A \succ \blacktriangleright BCD \rhd]$	$\{abcd\}$
5	$bdca = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$* * * \; d \; * * *$	$\{ab\}$	$* * * \; c \; * * *$	$\{a\}$	$[A \succ \blacktriangleright BCD \rhd]$	$\{abcd\}$
D	$bdac = \{\}$	$[\lhd A \blacktriangleleft \prec B]$	$\{a\}$	$* * * \; d \; * * *$	$\{ab\}$	$* * * \; a \; * * *$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
2	$cabd = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ B \blacktriangleright \rhd \prec C]$	$\{ab\}$	$[B \succ \blacktriangleright C \rhd]$	$\{abc\}$	D	$\{abcd\}$
E	$cadb = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ B \blacktriangleright \rhd \prec C]$	$\{ab\}$	$[B \succ \lhd \blacktriangleleft C \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD \rhd]$	$\{abcd\}$
4	$cbad = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ \lhd B \rhd \prec C]$	$\{a\}$	$[A \succ \blacktriangleright BC \rhd]$	$\{abc\}$	D	$\{abcd\}$
6	$cbda = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ \lhd B \rhd \prec C]$	$\{a\}$	$[A \succ \lhd \blacktriangleleft BC \prec D]$	$\{a\}$	$[A \succ \blacktriangleright BCD \rhd]$	$\{abcd\}$
7	$cdba = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ B, B \succ \lhd \blacktriangleleft C \prec D]$	$\{ab\}$	$[A \lhd B \rhd \prec C, C \prec D]$	$\{a\}$	$[A \succ \blacktriangleright BCD \rhd]$	$\{abcd\}$
F	$cdab = \{\}$	$[\lhd AB \blacktriangleleft \prec C]$	$\{ab\}$	$[A \succ B, B \succ \lhd \blacktriangleleft C \prec D]$	$\{ab\}$	$[A \succ B \blacktriangleright \rhd \prec C, C \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD \rhd]$	$\{abcd\}$
4	$dabc = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$[A \succ BC \blacktriangleright \rhd \prec D]$	$\{abc\}$	$[B \succ C \blacktriangleright \rhd \prec D]$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
6	$dacb = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$[A \succ BC \blacktriangleright \rhd \prec D]$	$\{abc\}$	$[B \succ \lhd C \rhd \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD]$	$\{abcd\}$
5	$dbac = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$* * * \; b \; * * *$	$\{ab\}$	$* * * \; a \; * * *$	$\{abc\}$	$[C \succ \blacktriangleright D \rhd]$	$\{abcd\}$
G	$dbca = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$* * * \; b \; * * *$	$\{ab\}$	$* * * \; c \; * * *$	$\{a\}$	$[A \blacktriangleright BCD \rhd]$	$\{abcd\}$
7	$dcab = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$[A \succ B, B \succ \lhd C \rhd \prec D]$	$\{ab\}$	$[A \succ B \blacktriangleright \rhd \prec C, C \prec D]$	$\{ab\}$	$[B \succ \blacktriangleright CD]$	$\{abcd\}$
H	$dcba = \{\}$	$[\lhd ABC \blacktriangleleft \prec D]$	$\{abc\}$	$[A \succ B, B \succ \lhd C \rhd \prec D]$	$\{ab\}$	$[A \succ \lhd B \rhd \prec C, C \prec D]$	$\{a\}$	$[A \succ \blacktriangleright BCD]$	$\{abcd\}$

Figure 9: Permutations of $ABCD$

we first have the patches b and d in some order, followed by the patches a and c in some order. By symmetry we know that the center context must be $\{ab\}$, but the conflictors that we create do not fit. For example, here is the sequence we create for the $DBCA$ case:

$$\begin{array}{l}
\{\}[\triangleleft ABC \blacktriangleleft \prec D]\{abc\} \\
\{ab\}[A \succ \triangleleft B \triangleright C \prec D]\{a\} \\
\{abc\}[A \succ B \triangleleft C \triangleright \prec D]\{ab\} \\
\{a\}[A \succ \blacktriangleright BCD \triangleright]\{abcd\}
\end{array}$$

As you can see, the contexts do not match. Thus it seems that our intuition is insufficient, and we need a different definition for conflictors; but what?

6 Glossary

A'	The patch with name a and effect A'
A'	The effect of A'
a	The patch name of A, A', ...
A	Some patch with name a; effect need not be A
A^{-1}	The inverse of A, where A can be a patch, patch effect or patch name
Es	The sequence Es, which may be of patches or patch effects
$[\![a; Es]\!]$	A conflictor with name a and effect Es
$\mathcal{E}(Es)$	The effect of the patch sequence Es
ϵ	The null effect
$\{abc\}$	The context $\{a, b, c\}$