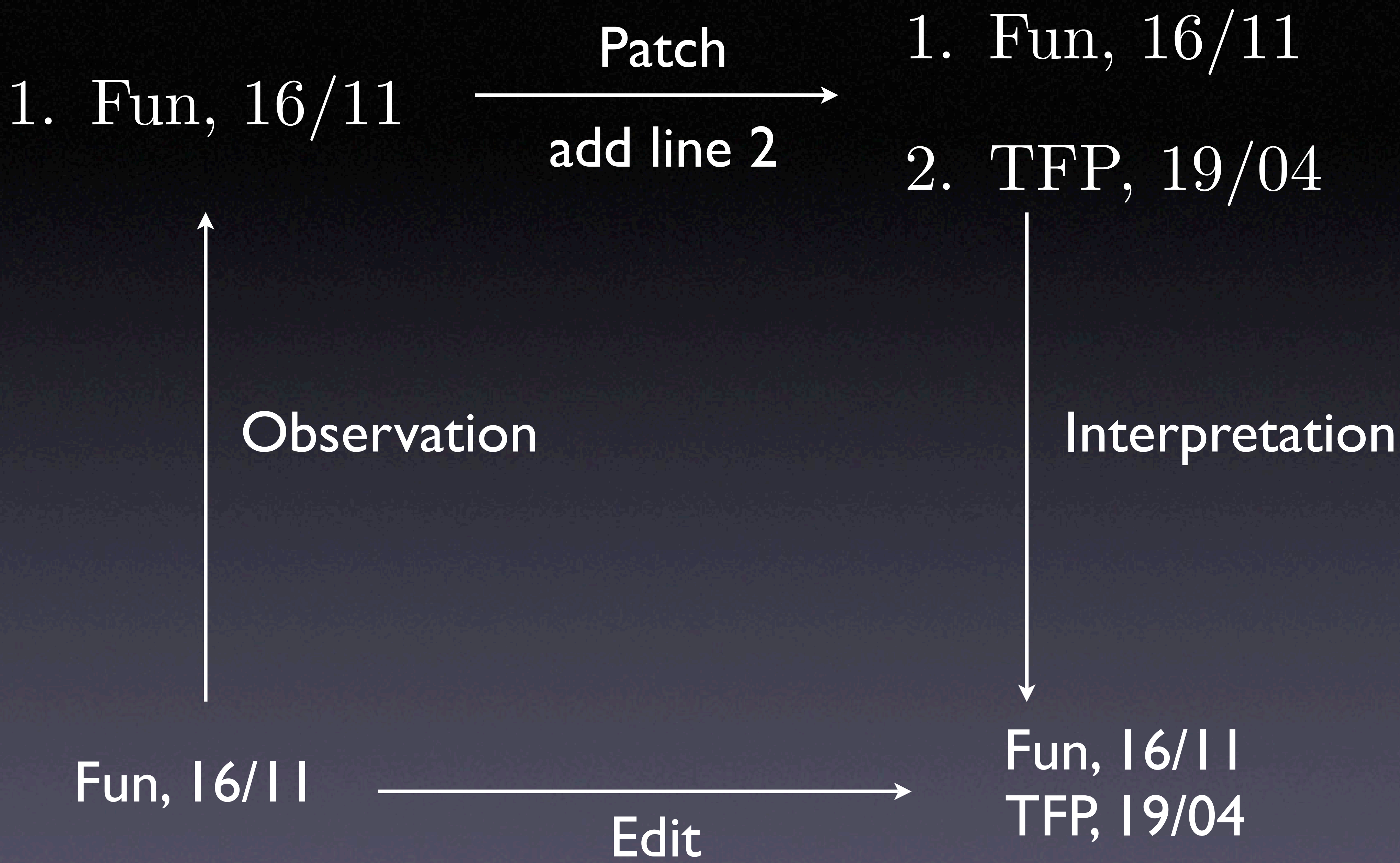
A Principled Approach to Version Control

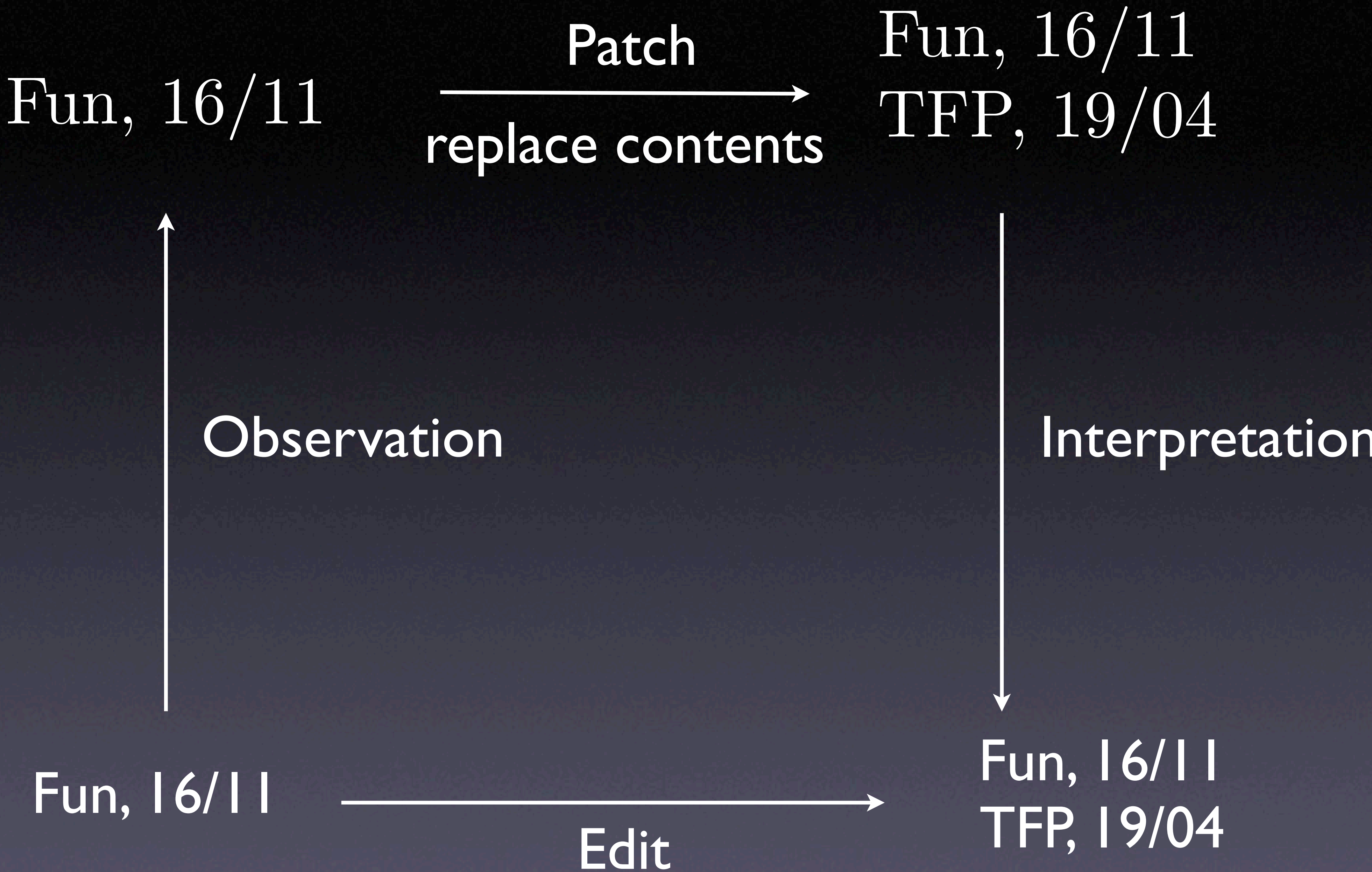
Wouter Swierstra

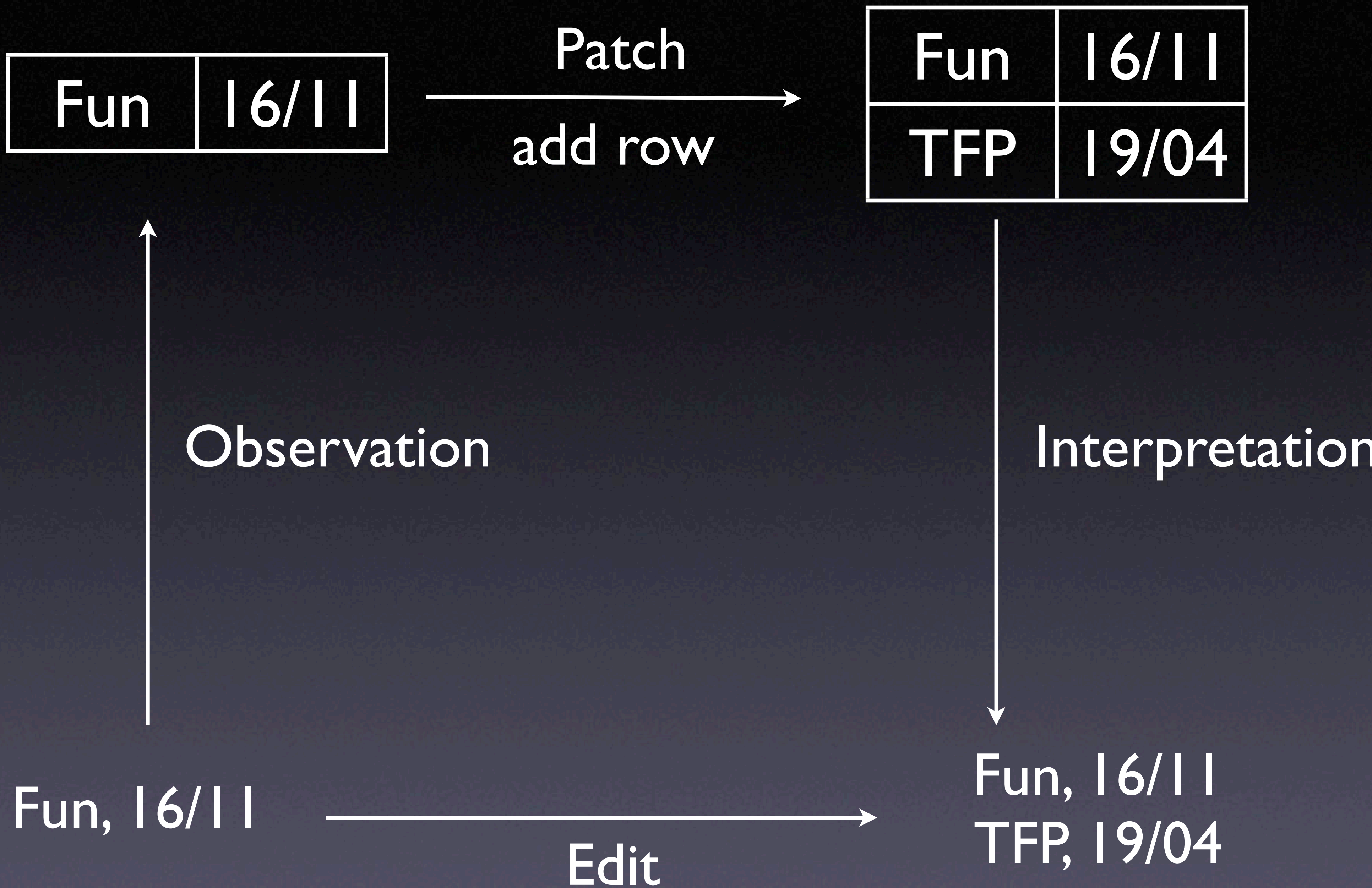
November 16, 2006

Version control is a real problem...

... and most tools are unpredictable.


1. Fun, 16/11
Patch
add line 2
1. Fun, 16/11
2. TFP, 19/04
Observation
Interpretation
Fun, 16/11
Edit
Fun, 16/11
TFP, 19/04


Fun, 16/11
Patch
replace contents
Fun, 16/11
TFP, 19/04
Observation
Interpretation
Fun, 16/11
Edit
Fun, 16/11
TFP, 19/04


Fun
16/11
Patch
add row
Fun
16/11
TFP
19/04
Observation
Interpretation
Fun, 16/11
Edit
Fun, 16/11
TFP, 19/04

Goal

A general theory of version control, abstracting over any possible design choice.

Example: binary files

- Let’s design a version control tool for managing binary files.
- What is a repository?
- What operations change the repository?

Internal Representation

- Suppose F is a set of file names.
- A repository is set of predicates:

$$f = c$$

which state that a file $f \in F$
has contents $c \in \text{Bits}$.

- Of course, we need to enforce an invariant:

$$\forall c, c' \in \text{Bits}.\, f = c \in R \wedge f = c' \in R \Rightarrow c = c'$$

Repository operations

- We want to allow three operations on repositories:

$$add\ f\ r = r \cup \{f = \varepsilon\}$$

$$delete\ f\ c\ r = r - \{f = c\}$$

$$modify\ f\ c\ d\ r = (r - \{f = c\}) \cup \{f = d\}$$

Why patches?

- Adding files may break the repository invariant.
- You can delete non-existing files.
- Reasoning about arbitrary functions can be arbitrarily difficult.
- Is there a general notion capable of describing all repository operations?

Simple patches

- A **simple patch** is a pair of sets, called the source and target respectively:

$$S \mapsto T$$

- Such a patch deletes S from the repository, and adds T
- To apply this patch to a repository, S must be present and $T - S$ must be absent.

Example patches

- Deleting a file

$$delete\ f\ c = \{f = c\} \mapsto \emptyset$$

- Modifying a file

$$modify\ f\ c\ d = \{f = c\} \mapsto \{f = d\}$$

- Adding a file

$$create\ f = \emptyset \mapsto \{f = \varepsilon\}$$

- This can still break repository invariants...

Invertible operations on points

- Present before, absent after.
- Present before, present after.
- Absent before, present after.
- Absent before, absent after.

Patches

- A **patch** is a triple of sets:

$$S \vdash E \rightarrow T$$

- Where E is a superset of both S and T
- A patch can be applied to a set X when

$$X \cap E = S$$

- We use E when some points must be absent.
- We still write $S \mapsto T$ when $S \cup T = E$

Creation revisited

- We can now define file creation as:

$$create\ f = \emptyset \vdash \{f = c \mid c \in \text{Bits}\} \to \{f = \varepsilon\}$$

- The extension guarantees that no existing file can be added to the repository
- Different design choices do exist, but now we now have the means to express them!

Patch composition

- Given simple patches $S \mapsto T$ and $T \mapsto U$ we build their composition:

$$S \vdash S \cup T \cup U \to U$$

- The general formula is a bit more complicated.
- Composition is associative.

Commutation and inverses

- All patches ‘commute’ in a certain sense.
- When $p_1 \cdot p_2$ and $p_2 \cdot p_1$ both exist and are applicable to X then
$$(p_1 \cdot p_2)(X) = (p_2 \cdot p_1)(X)$$
- Every patch $S \vdash E \to T$ has an inverse patch $T \vdash E \to S$

Beyond binary files

- Line based text files
- Directory structure
- File moves and renaming
- Structured data and structured operations
- Tagging versions
- Patch meta-data

Repositories

- A repository is a multiset of patches.
- A repository is consistent if its constituent patches can be composed and applied to the empty set.

Communicating change

- Give repositories R and S, a **pull** of a multiset $P \subseteq R$ to S consists of a multiset

$$P' \subseteq (R - S)$$

  such that $P \subseteq P'$ and $S \cup P'$ is a consistent repository.

- In general, we are only interested in minimal pulls.

Conflicts

- Sometimes there is no way to successful pull a desirable multiset of patches.
- Adding the patches is said to cause a **conflict.**
- A user is responsible for adding new patches, such that the repository is consistent once again.

Darcs

- One of the largest and most popular applications written in Haskell
- Darcs is great!
- Based on a theory of patches.

Theory of patches

- Rather vague at times
- Patches exist in a context.
- Commuting patches changes the patches:

$$AB \leftrightarrow B'A'$$

- Conflictors are special patches.
- Algebraic theory is quite difficult.

What's next?

- Explore the algebraic structure.
- Develop good algorithms.
- Implement ideas.