The Semantics of Version Control
Who here uses version control?
Who here is happy with the system they use?
Ask any seasoned developer about a long-delayed merge ... and watch the blood drain out of his or her face.
- Bryan O'Sullivan

1 Making Sense of Revision-Control Systems, Communications of the ACM, Vol. 52 No. 9, Pages 56-62
Version control systems

Version control systems like compilers for 'ancient' languages:

• they solve a hard problem
• but it's hard to predict their behaviour
• their design can be ad-hoc
• and they don't have a formal semantics.
What do version control systems do?
They manage access to mutable state.
There are logics for reasoning about this!
Terminology

Version control systems manage a repository, consisting of data stored on disk.

This data exists on two levels:

1. The *raw data* stored on disk;
2. The *internal model* of this data, managed by the VCS
Example

For example, most VCS treat have the following model:

- text files are a (linked) list of lines;
- binary files are blobs of bits;
- each file has permissions (which are tracked);
- but timestamps are ignored.
Back to programming languages

• A VCS's internal model is a 'heap'

• A patch is some change to a repositories internal model, that may shared between repositories.

• Patches modify the 'heap' - we should define their semantics using a suitable logic.
A trivial version control system

Define a version control system that tracks a single binary file.

• What is the internal model?
• What predicates can we formulate that observe properties of the model?
• What operations are there on this model?
Internal model

We define the type of our internal model $M$, assuming some valid set of file names $F$:

$$M := \epsilon \mid (F, \text{Bits})$$

$$\text{Bits} := (0|1)^*$$
Predicates

• If (the internal model of) the repository is \( (f, c) \) then we say the predicate \( f \mapsto c \) holds;

• If (the internal model of) the repository is \( \epsilon \) then we say the predicate \( \varnothing \) holds.

We will write \( M \models p \) when \( M \) satisfies the predicate \( p \).
Operations

We can define three operations that manipulate the repository as Hoare triples:

\[
\begin{align*}
\{\emptyset\} & \quad \text{create } f \quad \{f \mapsto \epsilon\} \\
\{f \mapsto c\} & \quad \text{replace } f \quad c \quad d \quad \{f \mapsto d\} \\
\{f \mapsto \epsilon\} & \quad \text{remove } f \quad \{\emptyset\}
\end{align*}
\]
Sequential composition
Conflicts

When applying a patch \( \{ P \} \circ \{ Q \} \) to a repository \( M \), for which \( M \not\models P \), we say that \( c \) causes a **conflict** in the repository \( M \).

This definition does not mention Alice and Bob.
What about multiple files?
Separation logic
Internal model & predicates

The internal model is a partial map from filenames to bits:

\[ M := F \rightarrow \text{Bits} \]

There are two predicates:

\[ M \models f \mapsto c \quad \text{iff} \quad M(f) = c \land \text{dom}(M) = \{f\} \]
\[ M \models \emptyset \quad \text{iff} \quad \text{dom}(M) = \emptyset \]
Operations

\[
\begin{align*}
\{\emptyset\} & \quad \text{add } f \quad \{f \mapsto \epsilon\} \\
\{f \mapsto \epsilon\} & \quad \text{remove } f \quad \{\emptyset\} \\
\{f \mapsto c\} & \quad \text{replace } f \ x \ c \ d \quad \{f \mapsto d\}
\end{align*}
\]
Separating conjunction

\[ M \models P \cdot Q \text{ holds iff} \]

we can partition \( M \) into two disjoint parts, \( M_0 \) and \( M_1 \), such that \( M_0 \models P \) and \( M_1 \models Q \).
The frame rule

\[
\begin{align*}
\{P\} & \ c \ \{Q\} \\
\{P \ast R\} & \ c \ \{Q \ast R\}
\end{align*}
\]

Provided \( c \) does not modify files mentioned by \( R \).

We need to prove soundness of the frame rule for our system.
Independence

Using the frame rule we can specify when two patches are independent - that is they modify different parts of the repository.

**Lemma:** Independent patches commute.

This formalizes the intuition that you can avoid conflicts by working on different files.
What else?

We can model:

- text files as linked lists of lines;
- (nested) directories;
- or even structured data, like CSV.

Using control flow, like conditionals, we can mimic branching and merging.
What next?

• Does it scale?
• Can define structure aware version control systems?
• Can we define an algebraic semantics that is sound with respect to the separation logic semantics?
Closure

As the fruits of programming-language research become more widely understood, programming is going to become a much more mathematical craft.

– John Reynolds
Closure

As the fruits of programming-language research become more widely understood, programming is going to become a much more mathematical craft.

– John Reynolds

We would love the same to be true of software development.