The Case Against Edit Scripts

Victor Cacciari Miraldo

December 30, 2019

1 Introduction

Edit scripts are bad. [**(1.1) Victor:**

- Too much redundancy implies expensive algorithms.
- Too restrictive on opeations implies not being able to duplicate or permute.
- When coupled with line-based diff, merges are bad.
- Show a couple examples.

]

We propose an extensional approach.

2 Background

[**(2.1) Victor:** Some edit-scripts; some about tree-diffing]

[**(2.2) Victor:** Primer on unification and substitution and term algebras]

3 Extensional Patches

Instead of linearizing trees and relying on very local operations such as insertion, deletions and copying of a single constructor, we can take the extensional look over patches and describe them by a mapping between sets of trees. Lets look at a simple patch that deletes the left subtree of a binary tree – which can be described by the $\mathit{Del}\ \mathit{Bin}\ (\mathit{Del}\ \ldots\ (\mathit{Cpy}\ \ldots\ \mathit{Nil}))$ edit script. A Haskell function that performs that operation can be given by:

$$
\begin{array}{l}
\mathit{delL}\ (\mathit{Bin}\ \_\ x) = \mathit{Just}\ x \\
\mathit{delL}\ \_ = \mathit{Nothing}
\end{array}
$$

The delL function specifies a domain – those trees with a Bin at their root – and a transformation, which forgets the root and its left child.

[**(3.1) Victor:** still deciding the order of examples here... this is messy; pardon]

Take the patch that swaps the children of a binary tree – which is already impossible to represent with edit-scripts. It could be represented by a Haskell function *swap*:

$$
\begin{aligned}
&swap\ (Bin\ x\ y) = Just\ (Bin\ y\ x)\\
&swap\ \_ = Nothing
\end{aligned}
$$

This *swap* function has a pattern, which identifies the domain of the function. In our case, we can only swap trees with a *Bin* constructor at the root. That is, *dom swap* is given by:

$$dom\ swap = \{Bin\ x\ y \mid x \in Tree, y \in Tree\}$$

[**(3.2) Victor:** Onto patches]

Definition 1. Let $\mathcal{T}_L$ be the term algebra for the language L augmented with a countable set V of variables. A patch $p = p_d \mapsto p_i$ consists in a pattern, p_d, and an expression, p_i — both elements of $\mathcal{T}_L$ — such that $\mathbf{vars}\ p_i \subseteq \mathbf{vars}\ p_d$. We sometimes refer to p_d and p_i as the deletion and insertion contexts of p.

Definition 2. We say an element $x \in \mathcal{T}_L$ is a *term* whenever $\mathbf{vars}\ x = \emptyset$.

The *swap* patch, for example, is represented by $Bin\ x\ y \mapsto Bin\ y\ x$, where x and y are taken from the set V of variables. Similarly to working with the λ-calculus, we assume variable names never clash between patches.

Definition 3. [**(3.3) Victor:** application] Let p be a patch over $\mathcal{T}_L$ and x a term over $\mathcal{T}_L$, we say p applies to x whenever p_d unifies with x. Let α be such substitution, the result of the application is $\alpha\ p_i$.

$$\mathbf{app}\ p\ x = y \iff \exists \alpha . \alpha\ x_d = \alpha\ x \wedge \alpha\ p_i = y$$

The identity patch is simply $x \mapsto x$.

Lemma 1. [***(3.4) Victor:*** *correctness of application*] *For all patch p and term x, if* $\mathbf{app}\ p\ x = y$ *then y is a term.*

Proof. todo □

This notion of application gives rise to an extensional equality of patches. We say patches p and q are equal, denoted $p \approx q$, whenever

$$\forall x.(\mathbf{app}\ p\ x = y \iff \mathbf{app}\ q\ x = z) \wedge y = z$$

It is easy to prove that $\approx$ above gives an equivalence relation.

Definition 4. [**(3.5) Victor:** composition] Let p and q be patches we say that p and q compose whenever p_d unifies with q_i — let σ be such mgu. Given two patches p and q that compose,
$p \circ q = sigma\ q_d \mapsto sigma\ p_i$

Lemma 2. [***(3.6) Victor:*** *composition is correct*] *Given p and q composable patches,* $\mathbf{app}\ (p \circ q)\ x = z$ *iff* $\mathbf{app}\ q\ x = y \wedge \mathbf{app}\ p\ y = z$.

Proof. transcribe from notebook □

Lemma 3. *For any patch p, the identity patch $x \mapsto x$ is a left and right identity to patch composition.*

Proof. trivial □

Lemma 4. *Given p and q composable patches, let $\sigma = \mathbf{mgu}(p_d, q_i)$, then there exists σ_p, σ_q such that $\sigma = \sigma_p \cup \sigma_q$ and $\sigma_p\ p_d = \sigma_q\ q_i$.*

Proof. Immediate since $\mathbf{vars}\ p \cap \mathbf{vars}\ q = \emptyset$. □

Lemma 5. *Given p and q composable patches, let $\sigma = \mathbf{mgu}(p_d, q_i)$, then σ is idempodent in q_d and p_i. That is, $\sigma\ \sigma\ q_d = \sigma\ q_d$ and similarly for p_i.*

Proof. transcribe □

With these lemmas at hand, we can prove associativity of our composition operator.

Lemma 6. *Let p and q be composable patches. Let r be a patch composable with $p \circ q$. Then, q and r are composable and p and $q \circ r$ are composable. Moreover, $(p \circ q) \circ r \approx p \circ (r \circ q)$*

Proof. transcribe from notebook; somewhat long. □