Best Title in the Universe

42

Victor Cacciari Miraldo Wouter Swierstra

University of Utrecht

{v.cacciarimiraldo,w.s.swierstra} at uu.nl

Abstract

stuff

Categories and Subject Descriptors D.1.1 [*look*]: for—this

General Terms Haskell

Keywords Haskell

1. Introduction

The majority of version control systems handle patches in a non-structured way. They see a file as a list of lines that can be inserted, deleted or modified, with no regard to the semantics of that specific file. The immediate consequence of such design decision is that we, humans, have to solve a large number of conflicts that arise from, in fact, non conflicting edits. Implementing a tool that knows the semantics of any file we happen to need, however, is no simple task, specially given the plethora of file formats we see nowadays.

This can be seen from a simple example. Lets imagine Alice and Bob are iterating over a cake's recipe. They decide to use a version control system and an online repository to keep track of their modifications.

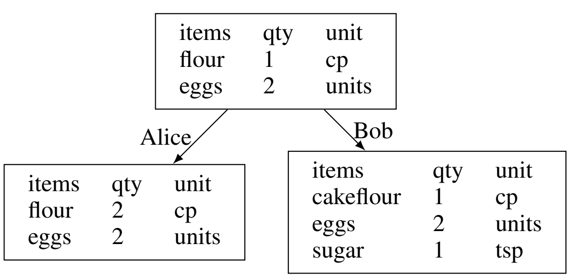

Figure 1. Sample CSV files

Lets say that both Bob and Alice are happy with their independent changes and want to make a final recipe. The standard way to track differences between files is the `diff3` [FIXBIB] unis tool. Running `diff3 Alice.csv O.csv Bob.csv` would result in the output presented in figure 2. Every tag `====` marks a difference. Three locations follows, formatted as `file:line type`. The change type can be a *Change*, *Append* or *Delete*. The first one, says that file 1 (`Alice.csv`) has a change in line 2 (`1:2c`) which is `flour, 2 , cp`; and files 2 and 3 have different changes in the same line. The tag `====3` indicates that there is a difference in file 3 only. Files 1 and 2 should append what changed in file 3 (line 4).

```
====
1:2c
  flour, 2  , cp
2:2c
  flour, 1  , cp
3:2c
  cakeflour, 1  , cp
====3
1:3a
2:3a
3:4c
  sugar, 1  , tsp
```

Figure 2. Output from `diff3`

If we try to merge the changes, `diff3` will flag a conflict and therefore require human interaction to solve it, as we can see by the presence of the `====` indicator in its output. However, Alice's and Bob's edits, in figure 1 do *not* conflict, if we take into account the semantics of CSV files. Although there is an overlapping edit at line 1, the fundamental editing unit is the cell, not the line.

We propose a structural diff that is not only generic but also able to track changes in a way that the user has the freedom to decide which is the fundamental editing unit. Our work is built on top of [FIXBIB], but we extend it in order to handle merging of patches. We also propose extensions to this algorithm capable of detecting purely structural operations such as swapping and cloning.

The paper begins by exploring the problem, generically, in the Agda [FIXBIB] language. Once we have a provably correct algorithm, the details of a Haskell implementation of generic diff'ing are sketched. To open ground for future work, we present a few extensions to our initial algorithm that could be able to detect semantical operations such as *cloning* and *swapping*.

Contributions

Background

- *Should we have this section? It cold be nice to at least mention the edit distance problem and that in the untyped scenario, the best running time is of $O(n^3)$. Types should allow us to bring this time lower.*

2. Structural Diffing

Alice and Bob were both editing a CSV file which represents data that is isomorphic to $[[Atom\ String]]$, where $Atom\ a$ is a simple tag that indicates that as should be treated as atomic.

- *What do we mean by structural?*
- *Give some context: Tree-edit distance;*
- *We seek to obtain a system with something close to residuals.*

$$\begin{tikzcd} & A_0 \arrow[dl, "p"'] \arrow[dr, "q"] & \\ A_1 \arrow[dr, "q/p"'] & & A_2 \arrow[dl, "p/q"] \\ & A_3 & \end{tikzcd}$$

To Research!

- *Check out the antidiagonal with more attention:* `http://blog.sigfpe.com/2007/09/type-of-distinct-pairs.html`
- *The LCS problem is closely related to diffing. We want to preserve the LCS of two structures! How does our diffing relate? Does this imply maximum sharing?*

2.1 Context Free Datatypes

- *Explain the universe we're using.*
- *Explain the intuition behing our D datatype.*
- *Mention that it is correct.*

```
data U : ℕ → Set where
  u0  : {n : ℕ} → U n
  u1  : {n : ℕ} → U n
  _⊕_ : {n : ℕ} → U n → U n → U n
  _⊗_ : {n : ℕ} → U n → U n → U n
  β   : {n : ℕ} → U (suc n) → U n → U n
  μ   : {n : ℕ} → U (suc n) → U n
  vl  : {n : ℕ} → U (suc n)
  wk  : {n : ℕ} → U n → U (suc n)
```

- *Explain the patching problem.*
- *We want a type-safe approach.*
- *Argue that the types resulting from our parser are in a sub-language of what we treated next.*
- *introduce* edit-script, diffing *and* patching *or* apply

2.2 Patches over a Context Free Type

- *Explain that a patch is something which we can apply.*
- *Loh's approach is too generic, as the diff function should have type $a \to a \to D\ a$.*

In order to simplify the presentation, we are gonna explicitely name variables and write our types in a more mathematical fashion, other than the Agda encoding. As we discussed earlier, a patch is an object that track differences in a given type. Different types will allow for different types of changes.

Definition 2.1 (Simple Patch). *We define a (simple) patch $D\ ty$ by induction on ty as:*

$$\begin{aligned} D\ 0 &= 0 \\ D\ 1 &= 1 \\ D\ (x \times y) &= D\ x \times D\ y \\ D\ (x + y) &= (D\ x + D\ y) + 2 \times (x \times y) \\ D\ (\mu X.F\ X) &= \mu X.(1 \\ &+\ D\ (F\ 1) \times X \\ &+\ 2 \times (F\ 1) \times X \\ &) \end{aligned}$$

Let's see the coproduct case in more detail. There are four different possibilities for the changes seen in a coproduct, just like there are four different combinations of constructors for two objects of type $Either\ a\ b$. The first and second options, namelly $D\ x$ and $D\ y$ track differences of a $Left\ a$ into a $Left\ a'$ and a $Right\ b$ into a $Right\ b'$, respectively. The other possibilities are representing a $Left\ a$ becoming a $Right\ b$ or vice-versa. The other branches are straight-forward.

Fixed Points

- *Very close to Vassena's and Andres approach;*
- *Explicit grow conflicts*

2.3 Sharing of Recursive Subterms

- If we want to be able to share recursive subexpressions we need a mutually recursive approach.

2.4 Remarks on Type Safety

- At which level of our design space we would like type-safety?
- Maybe after introducing the matrix idea it is clear that type-safety might be desirable only on the diff level, not on the patch level.

3. What About True Conflicts?

In order to track down conflicts we parametrize D over an abstract indexed family. This reveals a *free monad*-like structure and allows for in-place conflict resolution and tracking.

The actual type we use in Agda looks like

Note that the first constructor of D just asks for a suitably indexed A. With this in mind, we can start to define our residual operation.

```
_/_ : {n : ℕ}{t : Tel n}{ty : U n}
  → Patch t ty → Patch t ty → Maybe (D C t ty)

residual-symmetry-thm
  : {n : ℕ}{t : Tel n}{ty : U n}{k : D C t ty}
  → (d1 d2 : Patch t ty)
  → d1 / d2 ≡ just k
  → Σ (D C t ty → D C t ty)
    (λ op → d2 / d1 ≡ just (D-map C-sym (op k)))
residual-sym-stable : {n : ℕ}{t : Tel n}{ty : U n}{k : D C t ty}
  → (d1 d2 : Patch t ty)
  → d1 / d2 ≡ just k
  → forget <M> (d2 / d1) ≡ just (map (↓-map-↓ C-sym) (forget k))
```

4. Sketching a Control Version System

- Different views over the same datatype will give different diffs.

```
mutual
  data D {a}(A : {n : ℕ} → Tel n → U n → Set a) : {n : ℕ} → Tel n → U n → Set a where
    D-A    : {n : ℕ}{t : Tel n}{ty : U n} → A t ty → D A t ty

    D-void : {n : ℕ}{t : Tel n} → D A t u1
    D-inl  : {n : ℕ}{t : Tel n}{a b : U n}
           → D A t a → D A t (a ⊕ b)
    D-inr  : {n : ℕ}{t : Tel n}{a b : U n}
           → D A t b → D A t (a ⊕ b)
    D-setl  : {n : ℕ}{t : Tel n}{a b : U n}
            → ElU a t → ElU b t → D A t (a ⊕ b)
    D-setr  : {n : ℕ}{t : Tel n}{a b : U n}
            → ElU b t → ElU a t → D A t (a ⊕ b)
    D-pair : {n : ℕ}{t : Tel n}{a b : U n}
           → D A t a → D A t b → D A t (a ⊗ b)
    D-mu : {n : ℕ}{t : Tel n}{a : U (suc n)}
      → List (Dμ A t a) → D A t (μ a)
    D-β : {n : ℕ}{t : Tel n}{F : U (suc n)}{x : U n}
      → D A (tcons x t) F → D A t (β F x)
    D-top : {n : ℕ}{t : Tel n}{a : U n}
      → D A t a → D A (tcons a t) vl
    D-pop : {n : ℕ}{t : Tel n}{a b : U n}
      → D A t b → D A (tcons a t) (wk b)

  data Dμ {a}(A : {n : ℕ} → Tel n → U n → Set a) : {n : ℕ} → Tel n → U (suc n) → Set a where
    Dμ-A  : {n : ℕ}{t : Tel n}{a : U (suc n)} → A t (μ a) → Dμ A t a
    Dμ-ins : {n : ℕ}{t : Tel n}{a : U (suc n)} → ValU a t → Dμ A t a
    Dμ-del : {n : ℕ}{t : Tel n}{a : U (suc n)} → ValU a t → Dμ A t a
    Dμ-cpy : {n : ℕ}{t : Tel n}{a : U (suc n)} → ValU a t → Dμ A t a
    Dμ-dwn : {n : ℕ}{t : Tel n}{a : U (suc n)} → ValU a t → D A t (β a u1) → Dμ A t a
```

Figure 3. Complete Definition of D

- **newtype** annotations can provide a gread bunch of control over the algorithm.
- Directories are just rosetrees...

5. Related Work

- People have done similar things... or not.

6. Conclusion

- This is what we take out of it.