Patch Theory in Cubical Agda

Yeh, Yu-Chun

National Yang Ming Chiao Tung University
awooo.og@gmail.com

Abstract

This work aims to formalize homotopical patch theory [1] in Cubical Agda. We first formalize the elementary patch theory, including its repository type, patch interpreter, and merge operation, and then verify the reconciliation and symmetry properties of merge. Finally, we begin the formalization of patch theory with laws by defining the repository and implementing the basic swap action on repositories. We additionally define a rebase operation for the elementary patch theory and prove a consistency property relating rebase to merge, together with a symmetry property for rebase.

1 Background

Homotopy type theory [2] is an extension of Martin-Löf type theory that interprets identity types using ideas from homotopy theory. In HoTT, a type A can be veiwed as a space, terms $a, b : A$ correspond to points, and identity types $a \equiv b$ are represented by paths. Moreover, we can define path operations considered in HoTT, including identity paths $refl : a \equiv a$, inverse paths $!p : b \equiv a$ where $p : a \equiv b$, and composition of paths $q \circ p : a \equiv c$ where $p : a \equiv b$ and $q : b \equiv c$. Higher inductive types allow us to define a type with constructors not only for points but also for paths. We utilize higher inductive types to patch theories.

Patch theory is the theoretic study of version control systems, which considers how patches behave under operations such as composition, reverting, and merging. With a patch context representing a repository state, a patch represents a change to a repository. Patches are closed under identity, composition, and inverses. On top of that, patch laws refer to equalities between patches. Homotopical Patch Theory [1] represents patch theories as higher inductive types by considering points as patch contexts, paths as patches, and paths between paths as patch laws. The paper considers interpretations such as patch interpreters, which give abstract patch a concrete meaning, patch optimizers, which interpret patches into simpler but equivalent to origin, and patch histories, which interpret patches as a changelog rather than changes themselves. It also discusses merging.

In this work, we aim to use Cubical Agda to formalize the patch theories defined in the paper, including their definitions, interpreters, and merge operations. We also try to define rebase of patch theory and prove its properties.

2 Elementary Patch Theory

In elementary patch theory, we take a repository to be a single integer, and the patches to be operations that add to or subtract from it. Since any patches are applicable to any repository state, only one patch context is needed, called `num` here, and a patch can be represented as a path `num ≡ num`.

Then we define an interpreter that explains how patches work on a repository. Since in elementary patch theory, we treat the repository as an integer, we would like to interpret the patch context as $\mathbb{Z}$. We also interpret each patch as a bijection on $\mathbb{Z}$ because patches are

invertible. Since `add1` simply applies one more cycle to the repository, it can be interpreted as *successor*, where *successor* is a bijection on $\mathbb{Z}$ defined by $(\lambda x \to x + 1, \lambda x \to x - 1, \lambda x \to (x + 1) - 1 \equiv x, \lambda x \to (x - 1) + 1 \equiv x)$. With the interpreter, we can then prove properties of *identity*, *inversion*, and *composition* of patches.

After that, we can define the merge function. Merge is an operation that takes two patches (f_1, f_2) and returns two patches (g_1, g_2). It should satisfy the properties of *reconciliation* and *symmetry*, where *reconciliation* means

$$merge(f_1, f_2) = (g_1, g_2) \Longrightarrow g_1 \circ f_1 = g_2 \circ f_2$$

and *symmetry* means

$$merge(f_1, f_2) = (g_1, g_2) \Longrightarrow merge(f_2, f_1) \equiv (g_2, g_1).$$

Since, in elementary patch theory, there will not be any conflicts between patches, merge can be defined by simply swapping the patches, that is,

$$merge(f_1, f_2) \coloneqq (f_2, f_1).$$

We prove that it satisfies these properties by transferring patches into $\mathbb{Z}$.

In addition to the operations considered in homotopical patch theory, we extend the formalization by defining a rebase operation. Rebase takes two patches (f_1, f_2) and returns two patches (g_1, g_2). It makes g_1 the inverse of f_1, while $g_2 \circ f_2$ has the same result as the merge reconciliation. Again, since there will not be any conflicts and there is only one state `num` in elementary patch theory, we can simply define rebase as

$$rebase(f_1, f_2) = (f_1^{-1}, f_1).$$

For the properties, we prove that there is a consistency between rebase and merge, and that rebase also satisfies symmetry, that is,

$$rebase(f_1, f_2) = (g_1, g_2),\ rebase(f_2, f_1) = (h_1, h_2) \Longrightarrow g_2 \circ f_2 \equiv h_2 \circ f_1.$$

3 Patch Theory with Laws

In this patch theory, we treat the repository as a document with n lines of strings, with each patch being a swap operation called `_↔_[at]_`. We also define patch laws such as independence, which states that the order of patches that edit different lines should not affect the result, and no-operation, which states that swapping two identical strings will still result in the same document.

We stop at defining the interpreter for this theory. Since we have the repository as a document with n lines, it is reasonable for us to interpret the patch context as a `vector` that contains exactly n elements of type `string`. Based on the interpretation of the document, the patches can be interpreted as swapping strings on a specific line of a document. However, we have not yet finished defining the interpreter for both the independence and no-operation patch laws.

4 Future Work

It is possible for us to finish the remaining parts of the interpreter for the patch theory with laws. On top of that, we can continue by giving merge and rebase clear definitions and proving their properties.

A Definitions

Listing 1: elementary

```
data R : Type where
  num  : R
  add1 : num ≡ num
```

Listing 2: with-laws

```
data R {n : ℕ} : Type where
  doc : R
  ↭__[at]_ : (s1 s2 : String) → (i : Fin n) → doc ≡ doc
  indep : (s t u v : String) (i j : Fin n) → (i ≡ j → ⊥) → (↭__[at]_ {n = n
    } s t i) • (↭__[at]_ {n = n} u v j) ≡ (↭__[at]_ {n = n} u v j) • (↭__[at
    ]_ {n = n} s t i)
  noop : (s : String) (i : Fin n) → ↭__[at]_ {n = n} s s i ≡ refl
```

B Figures

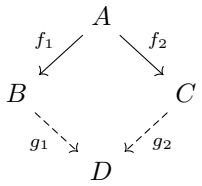


Figure 1: merge

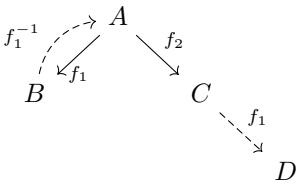


Figure 2: rebase

C Full Code

The code is available at GitHub.

References

[1] Carlo Angiuli, Edward Morehouse, Daniel R. Licata, and Robert Harper. Homotopical patch theory. In *Proceedings of the 19th ACM SIGPLAN International Conference on Functional Programming*, ICFP '14, pages 243–256, New York, NY, USA, August 2014. Association for Computing Machinery.

[2] The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics, August 2013.