A Cubical Implementation of Homotopical Patch Theory

Åsmund Aqissiaq Arild Kløvstad

Universitetet i Bergen

June 23rd, 2022

Overview

Homotopy Type Theory

Version Control Systems

Three Homotopical Patch Theories

1. An Elementary Patch Theory
2. A Patch Theory with laws
3. A Patch Theory with Richer Contexts

Computations

Homotopy Type Theory

Version Control Systems

Three Homotopical Patch Theories

1. An Elementary Patch Theory
2. A Patch Theory with laws
3. A Patch Theory with Richer Contexts

Computations

Types

- Types and terms

$$\mathsf{naturals} : \mathsf{Type}$$
$$\mathsf{naturals} = \mathbb{N}$$

$$\mathsf{aNumber} : \mathsf{naturals}$$
$$\mathsf{aNumber} = 0$$

- Elimination

$$\mathsf{double} : \mathbb{N} \to \mathbb{N}$$
$$\mathsf{double}\ \mathsf{zero} = \mathsf{zero}$$
$$\mathsf{double}\ (\mathsf{suc}\ n) = \mathsf{suc}\ (\mathsf{suc}\ (\mathsf{double}\ n))$$

- (Inductive) type families

```
data List (A : Type) : Type where
  []l : List A
  _::l_ : A → List A → List A
```

Dependent Types I

- A type that *depends* on a term

```
data Vec (A : Type) : ℕ → Type where
  [] : Vec A 0
  _::_ : ∀ {n} → A → Vec A n → Vec A (suc n)
```

- Dependent elimination

```
Vec-induction :
  {A : Type} → {P : ∀ {n} → Vec A n → Type} →
  (P []) →
  (∀ {n} → (a : A) → (as : Vec A n) → P (a :: as)) →
  {n : ℕ} → (v : Vec A n) → P v
```

Dependent Types II

$$\text{Vec-induction}\ \mathit{empty}\ \_\ [\,]\ =\ \mathit{empty}$$
$$\text{Vec-induction}\ \_\ \mathit{cons}\ (a :: v)\ =\ \mathit{cons}\ a\ v$$

- Π-types

$$\Pi : (X : \text{Type}) \to (X \to \text{Type}) \to \text{Type}$$
$$\Pi\ X\ P = (x : X) \to P\ x$$

$$\text{countDown} : \Pi\ \mathbb{N}\ (\text{Vec}\ \mathbb{N})$$
$$\text{countDown}\ \text{zero} = [\,]$$
$$\text{countDown}\ (\text{suc}\ x) = x :: (\text{countDown}\ x)$$

Dependent Types III

- Σ-types

```
record Σ (X : Type) (P : X → Type) : Type where
  field
    fst : X
    snd : P fst

_ : Σ ℕ (Vec ℕ)
_ = record { fst = 2 ; snd = 0 :: (1 :: []) }
```

Identity Types I

- When are two terms the same?

```
data Id {X : Type} (x : X) : X → Type where
  refl : Id x x

_ : Id (double 2) 4
_ = refl
```

Identity Types II

- The J rule (identity induction)

$$
\begin{array}{l}
\mathsf{J} : \{X : \mathsf{Type}\}\ \{x : X\} \to \\
\quad (P : (y : X) \to (\mathsf{Id}\ x\ y) \to \mathsf{Type}) \to \\
\quad (base : P\ x\ \mathsf{refl}) \to \\
\quad \overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \\
\quad (y : X) \to (p : \mathsf{Id}\ x\ y) \to P\ y\ p \\
\mathsf{J}\ P\ base\ y\ \mathsf{refl} = base
\end{array}
$$

Groupoids

- Identity types form an equivalence relation
- $\neg$UIP
- Groupoids (Hofman & Streicher '98)
  - identity (`refl`)
  - composition (transitivity)
  - inverses (symmetry)
- functions $\leftrightarrow$ functors

Higher Inductive Types I

- Custom groupoids
- Quotients (and truncation)

```
data _/_ (A : Type) (_∼_ : A → A → Type) : Type where
  [_] : A → A / _∼_
  eq : ∀ {a b} → a ∼ b → [ a ] ≡ [ b ]
  trunc : {a b : A / _∼_} → (p q : a ≡ b) → p ≡ q
```

- Synthetic Topology

```
data S¹ : Type where
  base : S¹
  loop : base ≡ base
```

Higher Inductive Types II

- Elimination

$$\begin{aligned}&\mathsf{reverse} : \mathsf{S}^1 \to \mathsf{S}^1\\&\mathsf{reverse}\ \mathsf{base} = \mathsf{base}\\&\mathsf{reverse}\ (\mathsf{loop}\ i) = \mathsf{loop}\ (\sim i)\end{aligned}$$

Setup

- Contexts
- Patches
- Patch operations
- Patch laws
- Patch *theories*

$$
\begin{tikzcd}
 & & C \\
A \arrow[r, "{}^{A}P^{B}"] \arrow[rrd, dashed, bend right, "{}^{A}(P\cdot R)^{D}"'] & B \arrow[ru, "{}^{B}Q^{C}"] \arrow[rd, "{}^{B}R^{D}"] \arrow[d, phantom, "\Vert\ comp"] & \\
 & & D
\end{tikzcd}
$$

Groupoid Structure

- Identity
- Composition
- Inverses

Homotopical Patch Theory (Angiuli, Morehouse, Licata, Harper 2014)

Take advantage of types' groupoid structure

VCS	HoTT	
patch theory	HIT	`R`
context	term	`doc : R`
patch	path	$\texttt{swap} : \texttt{doc} \equiv \texttt{doc}$
patch law	higher-order path	$\texttt{twice} : \texttt{swap} \cdot \texttt{swap} \equiv \texttt{refl}$
model	function	$\texttt{Interp} : \texttt{R} \to \texttt{Type}$

An Elementary Theory

- One context
- One patch
- Two models

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

```
ℤI : R → Type
ℤI num = ℤ
ℤI (patch i) = sucPathℤ i

BI : R → Type
BI num = Bool
BI (patch i) = notEq i
```

In Practice

```
_ : applyℤ patch 0 ≡ 1
_ = refl

_ : applyB patch true ≡ false
_ = refl

_ : applyℤ (patch · patch · (sym patch)) 0 ≡ 1
_ = refl
```

A Theory with Laws

- One context
- One patch
- Two patch laws

```
data R : Type where
  doc : R
  _↔_AT_ : String → String → Fin size → doc ≡ doc
  noop : ∀ s i → s ↔ s AT i ≡ refl
  indep : ∀ s t u v i j → i ≢ j →
    (s ↔ t AT i) · (u ↔ v AT j) ≡ (u ↔ v AT j) · (s ↔ t AT i)
```

Model

```
Interp : R → Type
Interp doc = Vec String size
Interp ((s ↔ t AT idx) i) = ua (swapat (s , t) idx) i
Interp (noop s i i₁ i₂) = {!!} -- swapat respects noop
Interp (indep s t u v i j x i₁ i₂) =
  {!!} -- swapat respects indep
```

A Patch Optimizer

- Program and prove

$$(p : \mathsf{doc} \equiv \mathsf{doc}) \to \Sigma[\ q \in \mathsf{doc} \equiv \mathsf{doc}\ ]\ p \equiv q$$

- Pointwise

$$\mathsf{opt} : (x : \mathsf{R}) \to \Sigma[\ y \in \mathsf{R}\ ]\ y \equiv x$$

- Then apply with λ `p` $\to$ `cong opt p`
- Patch laws are handled by contractibility of codomain

History

```
data History : ℕ → ℕ → Type where
  []        : {m : ℕ} → History m m
  ADD_AT_::_ : {m n : ℕ} (s : String) (l : Fin (suc n)) →
                  History m n → History m (suc n)
  RM_::_ : {m n : ℕ} (l : Fin (suc n)) →
                  History m (suc n) → History m n
```

The Theory

$$
\begin{aligned}
&\mathsf{data}\ \mathsf{R} : \mathsf{Type}\ \mathsf{where}\\
&\quad \mathsf{doc}\ \ : \{n : \mathbb{N}\} \to \mathsf{History}\ 0\ n \to \mathsf{R}\\
&\quad \mathsf{addP} : \{n : \mathbb{N}\}\ (s : \mathsf{String})\ (l : \mathsf{Fin}\ (\mathsf{suc}\ n))\\
&\qquad\qquad (h : \mathsf{History}\ 0\ n) \to \mathsf{doc}\ h \equiv \mathsf{doc}\ (\mathsf{ADD}\ s\ \mathsf{AT}\ l :: h)\\
&\quad \mathsf{rmP} : \{n : \mathbb{N}\}\ (l : \mathsf{Fin}\ (\mathsf{suc}\ n))\\
&\qquad\qquad (h : \mathsf{History}\ 0\ (\mathsf{suc}\ n)) \to \mathsf{doc}\ h \equiv \mathsf{doc}\ (\mathsf{RM}\ l :: h)
\end{aligned}
$$

Model

$$
\begin{aligned}
&\mathsf{replay} : \{n : \mathbb{N}\} \to \mathsf{History}\ 0\ n \to \mathsf{Vec\ String}\ n\\
&\mathsf{replay}\ [] = []\\
&\mathsf{replay}\ (\mathsf{ADD}\ s\ \mathsf{AT}\ l :: h) = \mathsf{add}\ s\ l\ (\mathsf{replay}\ h)\\
&\mathsf{replay}\ (\mathsf{RM}\ l :: h) = \mathsf{rm}\ l\ (\mathsf{replay}\ h)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{Interpreter} : \mathsf{R} \to \mathsf{Type}\\
&\mathsf{Interpreter}\ (\mathsf{doc}\ x) = \mathsf{singl}\ (\mathsf{replay}\ x)\\
&\mathsf{Interpreter}\ (\mathsf{addP}\ s\ l\ h\ i) =\\
&\quad \mathsf{ua}\ (\mathsf{singl\text{-}biject}\ \{a = \mathsf{replay}\ h\}\ (\mathsf{mapSingl}\ (\mathsf{add}\ s\ l)))\ i\\
&\mathsf{Interpreter}\ (\mathsf{rmP}\ l\ h\ i) =\\
&\quad \mathsf{ua}\ (\mathsf{singl\text{-}biject}\ \{a = \mathsf{replay}\ h\}\ (\mathsf{mapSingl}\ (\mathsf{rm}\ l)))\ i
\end{aligned}
$$

Setup

- with laws

```
repo : repoType
repo = "hello" :: "world" :: []

nop swap swap' comp : Patch
nop = "nop" ↔ "nop" AT (# 0)
swap = "hello" ↔ "greetings" AT (# 0)
swap' = "world" ↔ "earthlings" AT (# 1)
comp = swap · swap'
```

- with richer contexts

```
addPatch : doc [] ≡ doc (ADD "hello" AT zero :: [])
addPatch = addP "hello" zero []

rmPatch : doc (ADD "hello" AT zero :: []) ≡ doc (RM zero :: (ADD "hello" AT zero :: []))
rmPatch = rmP zero (ADD "hello" AT zero :: [])
```

transp

- with laws

```
_ : apply nop repo ≡ repo
_ = transportRefl repo

_ : apply swap repo ≡ "greetings" ∷ "world" ∷ []
_ = transportRefl _
```

- with richer contexts

```
_ : apply addPatch (S []) ≡ S ("hello" ∷ [])
_ = transportRefl _

_ : apply rmPatch (S ("hello" ∷ [])) ≡ S []
_ = transportRefl _

_ : apply rmPatch (apply addPatch (S [])) ≡ S []
_ = cong (apply rmPatch) (transportRefl ("hello" ∷ [] , refl))
    · (transportRefl ([] , refl))
```

hcomp

- with laws

```
_ : apply comp repo ≡ "greetings" :: "earthlings" :: []
_ = {!!}
```

- with richer contexts

```
_ : apply (addPatch · rmPatch) (S []) ≡ S []
_ = apply (addPatch · rmPatch) (S (replay []))
  ≡⟨ transportRefl _ ⟩ _
  ≡⟨ {!!} ⟩ S (replay (RM zero :: (ADD "hello" AT zero :: []))) ∎
```

The Curious Case of opt

```
nopOpt swapOpt compOpt : Patch
nopOpt = fst (optimize nop)
swapOpt = fst (optimize swap)
compOpt = fst (optimize comp)

-- _ : apply swapOpt repo ≡ "greetings" :: "world" :: []
-- _ = transportRefl "greetings" :: "world" :: []

-- _ : apply nopOpt repo ≡ repo
-- _ = transportRefl repo

-- _ : apply compOpt repo ≡ "greetings" :: "earthlings" :: []
-- _ = transportRefl _
```