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 and Challenges

Types

- Types and terms

$$\begin{aligned}&\text{naturals} : \text{Type}\\&\text{naturals} = \mathbb{N}\end{aligned}$$

$$\begin{aligned}&\text{aNumber} : \text{naturals}\\&\text{aNumber} = 3\end{aligned}$$

- Elimination

$$\begin{aligned}&\text{double} : \mathbb{N} \to \mathbb{N}\\&\text{double zero} = \text{zero}\\&\text{double } (\text{suc } n) = \text{suc } (\text{suc } (\text{double } n))\end{aligned}$$

- (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

$$
\begin{array}{l}
\text{data Vec } (A : \text{Type}) : \mathbb{N} \to \text{Type where} \\
\quad [] : \text{Vec } A \; 0 \\
\quad \_{::}\_ : \forall \, \{n\} \to A \to \text{Vec } A \; n \to \text{Vec } A \; (\text{suc } n)
\end{array}
$$

- Dependent elimination

$$
\begin{array}{l}
\text{map} : \forall \, \{A \; B \; n\} \to (A \to B) \to \text{Vec } A \; n \to \text{Vec } B \; n \\
\text{map } \_ \; [] = [] \\
\text{map } f \, (x :: xs) = f \, x :: \text{map } f \, xs
\end{array}
$$

Dependent Types II

- Π-types

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

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

Dependent Types III

- Σ-types

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

_ : Σ ℕ (Vec ℕ)
_ = 2 , (0 :: (1 :: []))
```

Identity Types I

- When are two terms the same?

$$\mathsf{data}\ \mathsf{Id}\ \{X : \mathsf{Type}\}\ (x : X) : X \to \mathsf{Type}\ \mathsf{where}$$
$$\quad \mathsf{refl} : \mathsf{Id}\ x\ x$$

$$\_ : \mathsf{Id}\ (\mathsf{double}\ 2)\ 4$$
$$\_ = \mathsf{refl}$$

Identity Types II

- The J rule (identity induction)

$$
\begin{aligned}
\mathsf{J} : \{X : \mathsf{Type}\}\ \{x : X\} \to\\
(P : (y : X) \to (\mathsf{Id}\ x\ y) \to \mathsf{Type}) \to\\
(\mathit{base} : P\ x\ \mathsf{refl}) \to\\
\hline
(y : X) \to (p : \mathsf{Id}\ x\ y) \to P\ y\ p
\end{aligned}
$$

- Some properties

$$
\begin{aligned}
&\mathsf{sym} : \mathsf{Id}\ x\ y \to \mathsf{Id}\ y\ x\\
&\mathsf{sym} = \mathsf{J}\ (\lambda\ y'\ p \to \mathsf{Id}\ y'\ x)\ \mathsf{refl}\ y
\end{aligned}
$$

$$
\begin{aligned}
&\_\cdot\_ : \mathsf{Id}\ x\ y \to \mathsf{Id}\ y\ z \to \mathsf{Id}\ x\ z\\
&\_\cdot\_\ p = \mathsf{J}\ (\lambda\ z'\ q \to \mathsf{Id}\ x\ z')\ p\ z
\end{aligned}
$$

Cubical Identity

- Paths

$$\mathsf{refl} : x \equiv x$$
$$\mathsf{refl} = \lambda\ i \to x$$

$$\mathsf{sym} : x \equiv y \to y \equiv x$$
$$\mathsf{sym}\ p = \lambda\ i \to p\ (\sim i)$$

- Composition

$$\begin{tikzcd}
x \arrow[r, dashed] \arrow[d, "p"'] & w \arrow[d, "r^{-1}"] \\
y \arrow[r, "q"'] & z
\end{tikzcd}$$

Groupoids

- $\neg$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

$$\mathsf{reverse} : S^1 \to S^1$$
$$\mathsf{reverse}\ \mathsf{base} = \mathsf{base}$$
$$\mathsf{reverse}\ (\mathsf{loop}\ i) = \mathsf{loop}\ (\tilde{}\ i)$$

Setup

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

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

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	`swap : doc` $\equiv$ `doc`
patch law	higher-order path	`twice : swap` $\cdot$ `swap` $\equiv$ `refl`
model	function	`Interp : R` $\rightarrow$ `Type`

An Elementary Theory

- One context
- One patch
- Two models

$$
\begin{aligned}
&\mathsf{data}\ \mathrm{R} : \mathrm{Type}\ \mathsf{where}\\
&\quad \mathrm{num} : \mathrm{R}\\
&\quad \mathrm{patch} : \mathrm{num} \equiv \mathrm{num}
\end{aligned}
$$

$$
\begin{aligned}
&\mathbb{Z}\mathrm{I} : \mathrm{R} \to \mathrm{Type}\\
&\mathbb{Z}\mathrm{I}\ \mathrm{num} = \mathbb{Z}\\
&\mathbb{Z}\mathrm{I}\ (\mathrm{patch}\ i) = \mathrm{sucPath}\mathbb{Z}\ i
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{BI} : \mathrm{R} \to \mathrm{Type}\\
&\mathrm{BI}\ \mathrm{num} = \mathrm{Bool}\\
&\mathrm{BI}\ (\mathrm{patch}\ i) = \mathrm{notEq}\ i
\end{aligned}
$$

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

$$
\begin{aligned}
&\mathsf{Interp} : \mathsf{R} \to \mathsf{Type}\\
&\mathsf{Interp}\ \mathsf{doc} = \mathsf{Vec}\ \mathsf{String}\ \mathsf{size}\\
&\mathsf{Interp}\ ((s \leftrightarrow t\ \mathsf{AT}\ idx)\ i) = \mathsf{ua}\ (\mathsf{swapat}\ (s\ ,\ t)\ idx)\ i\\
&\mathsf{Interp}\ (\mathsf{noop}\ s\ idx\ i\ j) = \mathsf{swapatNoop}\ s\ idx\ i\ j\\
&\mathsf{Interp}\ (\mathsf{indep}\ s\ t\ u\ v\ n\ m\ n{\neq}m\ i\ j) =\\
&\quad \mathsf{swapatIndep}\ s\ t\ u\ v\ n\ m\ n{\neq}m\ i\ j
\end{aligned}
$$

A Patch Optimizer

- Program and prove

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

- Pointwise

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

- Then apply with $\lambda\ \texttt{p} \to \texttt{cong opt p}$
- Patch laws are handled by contractibility of codomain

History

$$
\begin{array}{l}
\mathsf{data}\ \mathsf{History} : \mathbb{N} \to \mathbb{N} \to \mathsf{Type}\ \mathsf{where} \\
\quad [] \qquad\quad : \{m : \mathbb{N}\} \to \mathsf{History}\ m\ m \\
\quad \mathsf{ADD\_AT\_::\_} : \{m\ n : \mathbb{N}\}\ (s : \mathsf{String})\ (l : \mathsf{Fin}\ (\mathsf{suc}\ n)) \to \\
\qquad\qquad\qquad\qquad \mathsf{History}\ m\ n \to \mathsf{History}\ m\ (\mathsf{suc}\ n) \\
\quad \mathsf{RM\_::\_} : \{m\ n : \mathbb{N}\}\ (l : \mathsf{Fin}\ (\mathsf{suc}\ n)) \to \\
\qquad\qquad\qquad\qquad \mathsf{History}\ m\ (\mathsf{suc}\ n) \to \mathsf{History}\ m\ n
\end{array}
$$

The Theory

```
data R : Type where
  doc  : {n : ℕ} → History 0 n → R
  addP : {n : ℕ} (s : String) (l : Fin (suc n))
         (h : History 0 n) → doc h ≡ doc (ADD s AT l :: h)
  rmP  : {n : ℕ} (l : Fin (suc n))
         (h : History 0 (suc n)) → doc h ≡ doc (RM l :: h)
```

Model

```
replay : {n : ℕ} → History 0 n → Vec String n
replay [] = []
replay (ADD s AT l :: h) = add s l (replay h)
replay (RM l :: h) = rm l (replay h)

Interpreter : R → Type
Interpreter (doc x) = singl (replay x)
Interpreter (addP s l h i) =
  ua (singl-biject {a = replay h} (mapSingl (add s l))) i
Interpreter (rmP l h i) =
  ua (singl-biject {a = replay h} (mapSingl (rm l))) i
```

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 and Challenges

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 _
```

The `indep` Patch Law

- The program and prove approach relies on contractibility
- `noop`-"trick" (from set-truncation elimination rule)
- Non-terminating on `indep`

```
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)
```

```
opt (noop s j i k) = isOfHLevel→isOfHLevelDep 2
  (isProp→isSet ∘ isContr→isProp ∘ result-contractible)
  _ _ (cong opt (s ↔ s AT j)) refl (noop s j) i k
```

Another Approach I

- Coequalizers
- Binary path induction

$$\frac{\begin{array}{l}\ldots \\ \{x : A\} \to (p : [a0] \equiv [*]) \to \\ \{c : Maybe\} \to (q : [a0] \equiv [c]) \to P\ p\ q \\ \simeq (\{c : Maybe\}(q : [a0] \equiv [c]) \to P\ (p \cdot \mathrm{sett}\ \ x)\ q))\end{array}}{(p : [a0] \equiv [b]) \to (q : [a0] \equiv [c]) \to P\ p\ q}$$

Another Approach II

$$
\begin{tikzcd}
 & & z \arrow[dl, "p"'] \arrow[dr, "q"] & & \\
 & {[*]} \arrow[dl, "\text{sett } x"'] \arrow[dr, dashed] & & y \arrow[dl, dashed] \arrow[dr, dashed] & \\
{[x]} \arrow[ddrr, dashed] & & n \arrow[dd, phantom, "\wr"] & & y \arrow[ddll, dashed] \\
 & & & & \\
 & & n' & &
\end{tikzcd}
$$

Closing Thoughts

- Formalization with HITs and ua in Cubical Agda is fairly straight-forward
- transp/hcomp over inductive families is a hurdle
- Future work
  - More laws
  - Program *then* prove?
  - Investigate `optimize`
  - Directed paths?