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{array}{l} \text{naturals}: \ \text{Type} \\ \text{naturals} = \mathbb{N} \\ \text{aNumber}: \ \text{naturals} \\ \text{aNumber} = 3 \end{array}
```

Elimination

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

► (Inductive) type families

```
data List (A: Type): Type where []I: List A _::I_: A \rightarrow List A \rightarrow List A
```

Dependent Types I

► A type that *depends* on a term

```
data Vec (A : Type) : \mathbb{N} \to \text{Type where}

[] : Vec A 0

_::_ : \forall \{n\} \to A \to \text{Vec } A \ n \to \text{Vec } A \ (\text{suc } n)
```

► Dependent elimination

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

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 \Sigma (X: Type) (P: X \to \text{Type}): Type where constructor _,_ field  
   fst: X  
   snd: P fst  
_: \Sigma \mathbb{N} (Vec \mathbb{N})  
_= 2 , (0 :: (1 :: []))
```

Identity Types I

▶ When are two terms the same?

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

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

Identity Types II

► The J rule (identity induction)

Some properties

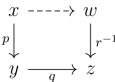
$$\begin{array}{l} \operatorname{sym}: \operatorname{Id} x \, y \to \operatorname{Id} y \, x \\ \operatorname{sym} = \operatorname{J} \left(\lambda \, y' \, p \to \operatorname{Id} \, y' \, x \right) \, \operatorname{refl} \, y \\ \\ \stackrel{\cdot}{\ldots}: \operatorname{Id} x \, y \to \operatorname{Id} \, y \, z \to \operatorname{Id} \, x \, z \\ \\ \stackrel{\cdot}{\ldots} \, p = \operatorname{J} \left(\lambda \, z' \, q \to \operatorname{Id} \, x \, z' \right) \, p \, z \end{array}$$

Cubical Identity

Paths

$$\begin{split} \text{refl} &: x \equiv x \\ \text{refl} &= \lambda \ i \rightarrow x \\ \\ \text{sym} &: x \equiv y \rightarrow y \equiv x \\ \text{sym} & p = \lambda \ i \rightarrow p \ (^{\sim} \ i) \\ \end{split}$$

Composition



Groupoids

- > ¬UIP
- Groupoids (Hofman & Streicher '98)
 - ▶ identity (refl)
 - composition (transitivity)
 - inverses (symmetry)
- ► Functions ↔ functors

Higher Inductive Types I

- Custom groupoids
- Quotients (and truncation)

```
data _/_ (A : Type) (_~_ : A \rightarrow A \rightarrow Type) : Type where [_] : A \rightarrow A / _~_ eq : \forall {a b} \rightarrow a ~ b \rightarrow [ a ] \equiv [ b ] trunc : {a b : A / _~_} \rightarrow (p q : a \equiv b) \rightarrow p \equiv q
```

Synthetic Topology

```
\begin{array}{l} \text{data } S^1: \text{ Type where} \\ \text{base : } S^1 \\ \text{loop : base} \equiv \text{base} \end{array}
```

Higher Inductive Types II

► Elimination

```
reverse : S^1 \rightarrow S^1
reverse base = base
reverse (loop i) = loop (\tilde{i})
```

Homotopy Type Theory

Version Control Systems

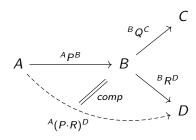
Three Homotopical Patch Theories

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

Computations and Challenges

Setup

- Contexts
- Patches
- Patch operations
- ► Patch laws
- ▶ Patch theories



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	$ extsf{swap}: extsf{doc} \equiv extsf{doc}$
patch law	higher-order path	$oxed{twice}$: $oxed{swap}$ $oxed{\cdot}$ $oxed{swap}$ $oxed{\equiv}$ $oxed{refl}$
model	function	$oxed{Interp:} exttt{R} ightarrow exttt{Type}$

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

An Elementary Theory

- One context
- ► One patch
- ► Two models

data R: Type where

num: R

patch : num ≡ num

 $\mathbb{Z}I:\mathsf{R}\to\mathsf{Type}$

 \mathbb{Z} I num = \mathbb{Z}

 $\mathbb{Z}I$ (patch i) = sucPath \mathbb{Z} i

 $\mathsf{BI}:\mathsf{R}\to\mathsf{Type}$

BI num = Bool

BI (patch i) = notEq i

```
Three Homotopical Patch Theories
```

└1. An Elementary Patch Theory

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

Three Homotopical Patch Theories

└2. A Patch Theory with laws

Model

```
Interp: R \rightarrow Type
Interp doc = Vec String size
Interp ((s \leftrightarrow t \text{ AT } idx) \ i) = ua \text{ (swapat } (s , t) \ idx) \ i
Interp (noop s \ idx \ i \ j) = swapatNoop \ s \ idx \ i \ j
Interp (indep s \ t \ u \ v \ n \ m \ n \neq m \ i \ j) = swapatIndep \ s \ t \ u \ v \ n \ m \ n \neq m \ i \ j
```

- Three Homotopical Patch Theories
 - └2. A Patch Theory with laws

A Patch Optimizer

Program and prove

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

Pointwise

opt :
$$(x : R) \rightarrow \Sigma[y \in R] y \equiv x$$

- ▶ Then apply with λ p \rightarrow cong opt p
- ▶ Patch laws are handled by contractibility of codomain

Three Homotopical Patch Theories

└3. A Patch Theory with Richer Contexts

History

```
data History: \mathbb{N} \to \mathbb{N} \to \mathsf{Type} where
[] \qquad : \{m : \mathbb{N}\} \to \mathsf{History} \ m \ m
\mathsf{ADD\_AT\_::}_:: \{m \ n : \mathbb{N}\} \ (s : \mathsf{String}) \ (I : \mathsf{Fin} \ (\mathsf{suc} \ n)) \to
\mathsf{History} \ m \ n \to \mathsf{History} \ m \ (\mathsf{suc} \ n)
\mathsf{RM\_::}_:: \{m \ n : \mathbb{N}\} \ (I : \mathsf{Fin} \ (\mathsf{suc} \ n)) \to
\mathsf{History} \ m \ (\mathsf{suc} \ n) \to \mathsf{History} \ m \ n
```

Three Homotopical Patch Theories

☐3. A Patch Theory with Richer Contexts

The Theory

```
data R: Type where
doc : \{n : \mathbb{N}\} \rightarrow \mathsf{History} \ 0 \ n \rightarrow \mathsf{R}
\mathsf{addP} : \{n : \mathbb{N}\} \ (s : \mathsf{String}) \ (I : \mathsf{Fin} \ (\mathsf{suc} \ n))
(h : \mathsf{History} \ 0 \ n) \rightarrow \mathsf{doc} \ h \equiv \mathsf{doc} \ (\mathsf{ADD} \ s \ \mathsf{AT} \ I :: \ h)
\mathsf{rmP} : \{n : \mathbb{N}\} \ (I : \mathsf{Fin} \ (\mathsf{suc} \ n))
(h : \mathsf{History} \ 0 \ (\mathsf{suc} \ n)) \rightarrow \mathsf{doc} \ h \equiv \mathsf{doc} \ (\mathsf{RM} \ I :: \ h)
```

- Three Homotopical Patch Theories
 - └3. A Patch Theory with Richer Contexts

Model

```
replay : \{n : \mathbb{N}\} \to \mathsf{History} \ 0 \ n \to \mathsf{Vec} \ \mathsf{String} \ n
replay [] = []
replay (ADD s AT I:: h) = add s I (replay h)
replay (RM I :: h) = rm I (replay h)
Interpreter : R \rightarrow Type
Interpreter (doc x) = singl (replay x)
Interpreter (addP s I h i) =
  ua (singl-biject \{a = \text{replay } h\} (mapSingl (add s l))) i
Interpreter (rmP I h i) =
  ua (singl-biject \{a = \text{replay } h\} (mapSingl (rm l))) i
```

Computations and Challenges

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" \( \to \) "nop" AT (# 0)
swap = "hello" \( \to \) "greetings" AT (# 0)
swap' = "world" \( \to \) "earthlings" AT (# 1)
comp = swap \( \to \) 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))
```

with laws

hcomp

```
-: 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

```
\begin{array}{l} \mathsf{noop} : \forall \ s \ i \to s \leftrightarrow s \ \mathsf{AT} \ i \equiv \mathsf{refl} \\ \mathsf{indep} : \forall \ s \ t \ u \ v \ i \ j \to i \not\equiv j \to \\ (s \leftrightarrow t \ \mathsf{AT} \ i) \cdot (u \leftrightarrow v \ \mathsf{AT} \ j) \equiv (u \leftrightarrow v \ \mathsf{AT} \ j) \cdot (s \leftrightarrow t \ \mathsf{AT} \ i) \\ \mathsf{opt} \ (\mathsf{noop} \ s \ j \ i \ k) = \mathsf{isOfHLevel} \to \mathsf{isOfHLevelDep} \ 2 \\ (\mathsf{isProp} \to \mathsf{isSet} \ \circ \ \mathsf{isContr} \to \mathsf{isProp} \ \circ \ \mathsf{result-contractible}) \\ -- (\mathsf{cong} \ \mathsf{opt} \ (s \leftrightarrow s \ \mathsf{AT} \ j)) \ \mathsf{refl} \ (\mathsf{noop} \ s \ j) \ i \ k \end{array}
```

Another Approach I

- Coequalizers
- ► Binary path induction

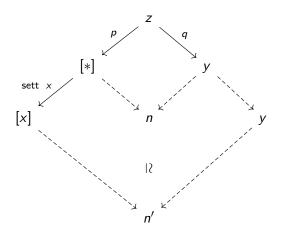
```
... \{x:A\} \rightarrow (p:[a0] \equiv [*]) \rightarrow

\{c:Maybe\} \rightarrow (q:[a0] \equiv [c]) \rightarrow P \ p \ q

\simeq (\{c:Maybe\}(q:[a0] \equiv [c]) \rightarrow P \ (p \cdot \text{sett} \ x) \ q))

(p:[a0] \equiv [b]) \rightarrow (q:[a0] \equiv [c]) \rightarrow P \ p \ q
```

Another Approach II



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?