A Cubical Implementation of Homotopical Patch Theory

Åsmund Aqissiaq Arild Kløvstad

Universitetet i Bergen

June 23rd, 2022

Overview

Types

- Types and terms

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

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

- Elimination

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

- (Inductive) type families

$$\text{data List } (A : \text{Type}) : \text{Type where}$$
$$\quad []\text{l} : \text{List } A$$
$$\quad \_{::}\text{l}\_ : A \to \text{List } A \to \text{List } A$$

Dependent Types I

- A type that *depends* on a term

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

- Dependent elimination

$$
\begin{aligned}
&\text{Vec-induction} : \\
&\quad \{A : \text{Type}\} \to \{P : \forall\ \{n\} \to \text{Vec } A\ n \to \text{Type}\} \to \\
&\quad (P\ [\,]) \to \\
&\quad (\forall\ \{n\} \to (a : A) \to (as : \text{Vec } A\ n) \to P\ (a :: as)) \to \\
&\quad \{n : \mathbb{N}\} \to (v : \text{Vec } A\ n) \to P\ v
\end{aligned}
$$

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?

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

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

Identity Types II

- The J rule (identity induction)

$$
\begin{array}{l}
\mathsf{J} : \{X : \mathsf{Type}\}\ \{x : X\} \to \\
\quad (P : (y : X) \to (\mathsf{Id}\ x\ y) \to \mathsf{Type}) \to \\
\quad (base : P\ x\ \mathsf{refl}) \to \\
\quad \overline{\hspace{20em}} \\
\quad (y : X) \to (p : \mathsf{Id}\ x\ y) \to P\ y\ p \\
\mathsf{J}\ P\ base\ y\ \mathsf{refl} = base
\end{array}
$$

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

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

Setup

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

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

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` $\to$ `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
```

```
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

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

- Pointwise

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

- Then apply with 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

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

Model

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

Setup

- with laws

$$\mathsf{repo} : \mathsf{repoType}$$
$$\mathsf{repo} = \texttt{"hello"} :: \texttt{"world"} :: []$$

$$\mathsf{nop}\ \mathsf{swap}\ \mathsf{swap}'\ \mathsf{comp} : \mathsf{Patch}$$
$$\mathsf{nop} = \texttt{"nop"} \leftrightarrow \texttt{"nop"}\ \mathsf{AT}\ (\#\ 0)$$
$$\mathsf{swap} = \texttt{"hello"} \leftrightarrow \texttt{"greetings"}\ \mathsf{AT}\ (\#\ 0)$$
$$\mathsf{swap}' = \texttt{"world"} \leftrightarrow \texttt{"earthlings"}\ \mathsf{AT}\ (\#\ 1)$$
$$\mathsf{comp} = \mathsf{swap} \cdot \mathsf{swap}'$$

- with richer contexts

$$\mathsf{addPatch} : \mathsf{doc}\ [] \equiv \mathsf{doc}\ (\mathsf{ADD}\ \texttt{"hello"}\ \mathsf{AT}\ \mathsf{zero} :: [])$$
$$\mathsf{addPatch} = \mathsf{addP}\ \texttt{"hello"}\ \mathsf{zero}\ []$$

$$\mathsf{rmPatch} : \mathsf{doc}\ (\mathsf{ADD}\ \texttt{"hello"}\ \mathsf{AT}\ \mathsf{zero} :: []) \equiv \mathsf{doc}\ (\mathsf{RM}\ \mathsf{zero} :: (\mathsf{ADD}\ \texttt{"hello"}\ \mathsf{AT}\ \mathsf{zero} :: []))$$
$$\mathsf{rmPatch} = \mathsf{rmP}\ \mathsf{zero}\ (\mathsf{ADD}\ \texttt{"hello"}\ \mathsf{AT}\ \mathsf{zero} :: [])$$

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

$$
\begin{tikzcd}
& & z \arrow[dl, "p"'] \arrow[dr, "q"] & & \\
& {[*]} \arrow[dl, "\mathsf{sett}\ x"'] \arrow[dr, dashed] & & y \arrow[dl, dashed] \arrow[dr, dashed] & \\
{[x]} \arrow[drr, dashed] & & n \arrow[d, phantom, "\wr"] & & y \arrow[dll, dashed] \\
& & n' & &
\end{tikzcd}
$$

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?