Git as a HIT

Dan Licata
Wesleyan University
Darcs

Git as a HIT

Dan Licata
Wesleyan University
HITs
HITs

*Homotopy Type Theory* is an extension of Agda/Coq based on connections with homotopy theory

[Hofmann&Streicher, Awodey&Warren, Voevodsky, Lumsdaine, Garner&van den Berg]
**HITs**

- *Homotopy Type Theory* is an extension of Agda/Coq based on connections with homotopy theory
  
  [Hofmann&Streicher, Awodey&Warren, Voevodsky, Lumsdaine, Garner&van den Berg]

- *Higher inductive types (HITs) are a new type former!"
HITs

- *Homotopy Type Theory* is an extension of Agda/Coq based on connections with homotopy theory
  
  [Hofmann&Streicher,Awodey&Warren,Voevodsky,Lumsdaine,Garner&van den Berg]

- *Higher inductive types (HITs)* are a new type former!

- They were originally invented [Lumsdaine,Shulman,...] to model basic spaces (circle, spheres, the torus, ...) and constructions in homotopy theory
HITs

- *Homotopy Type Theory* is an extension of Agda/Coq based on connections with homotopy theory
  
  [Hofmann&Streicher, Awodey&Warren, Voevodsky, Lumsdaine, Garner&van den Berg]

- *Higher inductive types (HITs)* are a new type former!

- They were originally invented [Lumsdaine, Shulman, …] to model basic spaces (circle, spheres, the torus, …) and constructions in homotopy theory

- But they have many other applications, including some programming ones!
Patches

diff

Patch

a b c

a d c

2c2

< b

---

> d
Simple Setup

\[
a \leftrightarrow b \text{ at } 0
\]
“Repository” is a char vector of fixed length n

Basic patch is \( a \leftrightarrow b \) at \( i \) where \( i < n \)
Domain-Specific Language

data Patch : Set where
  id       : Patch
  _°_      : Patch → Patch → Patch
  !        : Patch → Patch
  _↔_at_   : Char → Char → Fin n → Patch
Domain-Specific Language

interp : Patch → (Vec Char n → Vec Char n) × (Vec Char n → Vec Char n)

interp id = (λ x → x), (λ x → x)

interp (q ∘ p) = fst (interp q) o fst (interp p),
                 snd (interp p) o snd (interp q)

interp (! p) = snd (interp p), fst (interp p)

interp (a ↔ b at i) = swapat a b i, swapat a b i
Domain-Specific Language

\[ \text{interp} : \text{Patch} \rightarrow (\text{Vec Char} \ n \rightarrow \text{Vec Char} \ n) \times (\text{Vec Char} \ n \rightarrow \text{Vec Char} \ n) \]

\[ \text{interp} \ \text{id} = (\lambda \ x \rightarrow \ x), (\lambda \ x \rightarrow \ x) \]

\[ \text{interp} \ (q \circ p) = \text{fst} \ (\text{interp} \ q) \circ \text{fst} \ (\text{interp} \ p), \]
\[ \quad \text{snd} \ (\text{interp} \ p) \circ \text{snd} \ (\text{interp} \ q) \]

\[ \text{interp} \ (! \ p) = \text{snd} \ (\text{interp} \ p), \text{fst} \ (\text{interp} \ p) \]

\[ \text{interp} \ (a \leftrightarrow b \ \text{at} \ i) = \text{swapat} \ a \ b \ i, \text{swapat} \ a \ b \ i \]

\[ \text{swapat} \ a \ b \ i \ \forall \ \text{permutes} \ a \ \text{and} \ b \ \text{at position} \ i \ \text{in} \ v \]
Domain-Specific Language

\textbf{Spec:} \( \forall \ p. \ \text{interp} \ p \ \text{is a bijection:} \)
\[ \forall \ v. \ g (f \ v) = v \quad \text{where} \ (f,g) = \text{interp} \ p \]
\[ \forall \ v. \ f (g \ v) = v \]
Domain-Specific Language

Spec: ∀ p. interp p is a bijection:
  ∀ v. g (f v) = v where (f,g)=interp p
  ∀ v. f (g v) = v

undo really un-does
Domain-Specific Language

Spec: \( \forall p. \text{interp } p \text{ is a bijection:} \)
\( \forall v. g (f v) = v \) where \( (f,g) = \text{interp } p \)
\( \forall v. f (g v) = v \)

Can package this as:
\[
\text{interp} : \text{Patch} \rightarrow \text{Bijection (Vec Char n) (Vec Char n)}
\]
Merging

Diagram showing a merging process with elements a, b, c, p, q, d, and e.
Merging
Merging

\[\text{p} = b \leftrightarrow d \text{ at 1} \]
\[\text{q} = c \leftrightarrow e \text{ at 2} \]
Merging

$p = b \leftrightarrow d \text{ at } 1$
$q = c \leftrightarrow e \text{ at } 2$

$p' = p$
$q' = q$
Merging

$p = b \leftrightarrow d$ at 1
$q = c \leftrightarrow e$ at 2

$p' = p$
$q' = q$
Merging

\[
\text{merge : (p q : Patch)} \\
\rightarrow \Sigma q', p': \text{Patch}.
\]

\[
\text{Maybe}(q' \circ p = p' \circ q)
\]
Merging

merge : (p q : Patch)
→ ∑q’,p’:Patch.
Maybe(q’ o p = p’ o q)

When are two patches equal?
Patch Equality

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i) \text{ if } i \neq j\]
Patch Equality

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) =
\]
\[(c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i) \text{ if } i \neq j\]

\[(a \leftrightarrow a \text{ at } i) = id\]

\![a\leftrightarrow b \text{ at } i] = (a \leftrightarrow b \text{ at } i)\]

\[(a \leftrightarrow b \text{ at } i) = (b \leftrightarrow a \text{ at } i)\]
Patch Equality

**Basic Axioms:**

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i) \text{ if } i \neq j\]

\[(a \leftrightarrow a \text{ at } i) = id\]

\[!(a \leftrightarrow b \text{ at } i) = (a \leftrightarrow b \text{ at } i)\]

\[(a \leftrightarrow b \text{ at } i) = (b \leftrightarrow a \text{ at } i)\]
Patch Equality

**Basic axioms:**

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)\]
Patch Equality

**Basic axioms:**

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)\]

**Group laws:**

\[\text{id } \circ p = p = p \circ \text{id} \]
\[p \circ (q \circ r) = (p \circ q) \circ r\]
\[!p \circ p = \text{id } = p \circ !p\]
Patch Equality

Basic axioms:

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)\]

Congruence:

\[p = p\]
\[p = q \text{ if } q = p\]
\[p = r \text{ if } p = q \text{ and } q = r\]

Group laws:

\[\text{id } \circ p = p = p \circ \text{id}\]
\[p \circ (q \circ r) = (p \circ q) \circ r\]
\[\neg p \circ p = \text{id } = p \circ \neg p\]
\[\neg p = \neg p' \text{ if } p = p'\]
\[p \circ q = p' \circ q' \text{ if } p = p' \text{ and } q = q'\]
Patch as Quotient Type

Elements:

```haskell
data Patch' : Set where
  id    : Patch'
  id o id : Patch' 
  _~_ : Patch' → Patch' → Patch'
  !~ : Patch' → Patch'
  _~at_ : Char → Char → Fin n → Patch'
```

Equality:

```haskell
(a~b at i)o(c~d at j)~
  (c~d at j)o(a~b at i)

...
  id o p ~ p ~ p o id
  po(qor) ~ (poq)or
  !p o p ~ id ~ p o !p
  p~p
  p~q if q~p
  p~r if p~q and q~r
  !p ~ !p' if p ~ p'
  p o q ~ p' o q' if p ~ p' and q ~ q'
```
Patch as Quotient Type

Elements:

data Patch' : Set where
  id : Patch'
  _°_ : Patch' → Patch' → Patch'
  !_ : Patch' → Patch'
  _°_-at_- : Char → Char → Fin n → Patch'

Equality:

(a↔b at i)o(c↔d at j)~
  (c↔d at j)o(a↔b at i)

... id o p ~ p ~ p o id
  po(qor) ~ (poq)or
  !p o p ~ id ~ p o !p
  p~p
  p~q if q~p
  p~r if p~q and q~r
  !p ~ !p' if p ~ p'
  p o q ~ p' o q' if p ~ p' and q ~ q'

Quotient Type:

Patch := Patch’/~
Patch as Quotient Type

Elements:

\[
\begin{align*}
\text{data } \text{Patch'} : \text{Set} & \text{ where} \\
id : \text{Patch'} \\
\circ : \text{Patch'} \to \text{Patch'} \to \text{Patch'} \\
! : \text{Patch'} \to \text{Patch'} \\
\circ_\text{at} : \text{Char} \to \text{Char} \to \text{Fin} n \to \text{Patch'}
\end{align*}
\]

Equality:

\[
(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) \sim \\
(c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)
\]

\[
\cdots \\
id \circ p \sim p \circ id \\
po(qor) \sim (poq)or \\
!p \circ p \sim id \sim p \circ !p \\
p \sim p \\
p \sim q \text{ if } q \sim p \\
p \sim r \text{ if } p \sim q \text{ and } q \sim r \\
!p \sim !p' \text{ if } p \sim p' \\
p \circ q \sim p' \circ q' \text{ if } p \sim p' \text{ and } q \sim q'
\]

Quotient Type:

\[
\text{Patch} := \text{Patch'} / \sim
\]

Elimination rule:

\[
\text{interp : Patch } \to \\
\text{Bijection (Vec Char n) (Vec Char n)}
\]

define on Patch’ as before, then prove \( p \sim q \) implies 
\[
\text{interp } p = \text{interp } q
\]

for all 14+ rules for \( \sim \)
Patches as a HIT

1. How do you define Patch using a higher inductive type?

2. What is the elimination rule?

3. How do you use the elim. rule to define interp?
Patches as a HIT

1. How do you define \textit{Patch} using a higher inductive type?

2. What is the elimination rule?

3. How do you use the elim. rule to define interp?
Higher Inductive Type
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...

RepoDesc : Type
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...

RepoDesc : Type

vec : RepoDesc

generator for element
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...

RepoDesc : Type

vec : RepoDesc

(a\leftrightarrow b \text{ at i}) : \text{vec = vec}

(generator for element  generator for equality)
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...

RepoDesc : Type

vec : RepoDesc

(a ← b at i) : vec = vec

proof-relevant!

generator for element
generator for equality
Higher Inductive Type

Type freely generated by constructors for elements, equalities, equalities between equalities, ...

RepoDesc : Type

vec : RepoDesc

(a ↔ b at i) : vec = vec

commute:
(a ↔ b at i) o (c ↔ d at j) = (c ↔ d at j) o (a ↔ b at i)
Elements:

\[
\begin{align*}
id & : \text{Patch} \\
\circ & : \text{Patch} \to \text{Patch} \to \text{Patch} \\
! & : \text{Patch} \to \text{Patch} \\
\_\leftrightarrow \text{at} & : \text{Char} \to \text{Char} \to \text{Fin} \ n \to \text{Patch}
\end{align*}
\]

Equality:

\[
(a\leftrightarrow b \text{ at } i)o(c\leftrightarrow d \text{ at } j) = \\
(c\leftrightarrow d \text{ at } j)o(a\leftrightarrow b \text{ at } i)
\]

\[
\begin{align*}
id \circ p &= p = p \circ id \\
po(qor) &= (poq)or \\
!p \circ p &= id = p \circ !p \\
p &= p \\
p &= q \text{ if } q = p \\
p &= r \text{ if } p = q \text{ and } q = r \\
!p &= !p' \text{ if } p = p' \\
p \circ q &= p' \circ q' \text{ if } p = p' \text{ and } q = q'
\end{align*}
\]
**Type:** Patch

**Elements:**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Patch</td>
</tr>
<tr>
<td><em>⊙</em></td>
<td>Patch → Patch → Patch</td>
</tr>
<tr>
<td>!</td>
<td>Patch → Patch</td>
</tr>
<tr>
<td><em>↔</em>&lt;at&gt;</td>
<td>Char → Char → Fin n → Patch</td>
</tr>
</tbody>
</table>

**Equality:**

\[ (a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i) \]

... 

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>id o p = p = p o id</td>
<td></td>
</tr>
<tr>
<td>po(qor) = (poq)or</td>
<td></td>
</tr>
<tr>
<td>!p o p = id = p o !p</td>
<td></td>
</tr>
<tr>
<td>p=p</td>
<td></td>
</tr>
<tr>
<td>p=q if q=p</td>
<td></td>
</tr>
<tr>
<td>p=r if p=q and q=r</td>
<td></td>
</tr>
<tr>
<td>!p = !p’ if p = p’</td>
<td></td>
</tr>
<tr>
<td>p o q = p’ o q’ if p = p’ and q = q’</td>
<td></td>
</tr>
</tbody>
</table>
Type: Patch

Elements:

- `id` : Patch
- `_o_` : Patch → Patch → Patch
- `!` : Patch → Patch
- `_o_at_` : Char → Char → Fin n → Patch

Equality:

(a→b at i)o(c→d at j) =
(c→d at j)o(a→b at i)

... id o p = p = p o id
po(qor) = (poq)or
!p o p = id = p o !p
p=p
p=q if q=p
p=r if p=q and q=r
!p = !p' if p = p'
p o q = p' o q' if p = p' and q = q'
**Type:** Patch

### Elements:

- `id` : Patch
- `ο` : Patch → Patch → Patch
- `!` : Patch → Patch
- `→ at_` : Char → Char → Fin n → Patch

### Equality:

\[(a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j) = \]
\[= (c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } i)\]

\[...
\]

\[id \circ p = p = p \circ id\]

\[po(qor) = (poq)or\]

\![p \circ p = id = p \circ !p\]

\[p=p\]

\[p=q \text{ if } q=p\]

\[p=r \text{ if } p=q \text{ and } q=r\]

\[!p = !p' \text{ if } p = p'\]

\[p \circ q = p' \circ q' \text{ if } p = p' \text{ and } q = q'\]

**Type:** RepoDesc

**Element:** vec : RepoDesc

**Equality:**

\[a \leftrightarrow b \text{ at } i : \text{ vec } = \text{ vec}\]
Type: Patch

Elements:

- id : Patch
- _⊙_ : Patch → Patch → Patch
- !_ : Patch → Patch
- _← at_ : Char → Char → Fin n → Patch

Equality:

(a←b at i)o(c←d at j)=
(c←d at j)o(a←b at i)

... id o p = p = p o id
po(qor) = (poq)or
!p o p = id = p o !p
p=p
p=q if q=p
p=r if p=q and q=r
!p = !p' if p = p'
p o q = p' o q' if p = p' and q = q'
Type: Patch

Elements:

- id : Patch
- o : Patch → Patch → Patch
- ! : Patch → Patch
- <-at- : Char → Char → Fin n → Patch

Equality:

(a←b at i) o (c←d at j) =
(c←d at j) o (a←b at i)

... basic axioms only!

Type: RepoDesc

Element: vec : RepoDesc

Equality:

a←b at i : vec = vec

Equality between equalities:

commute :
(a←b at i) o (c←d at j) =
(c←d at j) o (a←b at i)

... basic axioms only!
Elements:

- `id : Patch`
- `\_\_\_ : Patch \rightarrow Patch \rightarrow Patch`
- `! : Patch \rightarrow Patch`
- `\_\_\_\_at_ : Char \rightarrow Char \rightarrow Fin n \rightarrow Patch`

Equality:

\[(a \leftrightarrow b \ at \ i) o (c \leftrightarrow d \ at \ j) = (c \leftrightarrow d \ at \ j) o (a \leftrightarrow b \ at \ i)\]

... basic axioms only!

```
  id o p = p = p o id
  po(qor) = (poq)or
  !p o p = id = p o !p
  p=p
  p=q if q=p
  p=r if p=q and q=r
  !p = !p' if p = p'
  p o q = p' o q' if p = p' and q = q'
```
Typed Patches

RepoDesc : Type

vec : RepoDesc
compressed : RepoDesc

\( a \leftrightarrow b \text{ at } i : \text{vec} = \text{vec} \)

gzip : \text{vec} = \text{compressed}
Typed Patches

RepoDesc : Type
vec : RepoDesc
compressed : RepoDesc

$a \leftrightarrow b$ at $i : vec = vec$
gzip : vec = compressed

Patch vec compressed
Patches as a HIT

1. How do you define Patch using a higher inductive type?

2. What is the elimination rule for RepoDesc?

3. How do you use the elim. rule to define interp?
RepoDesc recursion

To define a function $\text{RepoDesc} \rightarrow A$
it suffices to
To define a function \( \text{RepoDesc} \rightarrow A \)

it suffices to

- map the element generators of \( \text{RepoDesc} \)
  to elements of \( A \)
RepoDesc recursion

To define a function $\text{RepoDesc} \rightarrow A$ it suffices to

- map the element generators of $\text{RepoDesc}$ to elements of $A$
- map the equality generators of $\text{RepoDesc}$ to equalities between the corresponding elements of $A$
RepoDesc recursion

To define a function $\text{RepoDesc} \rightarrow A$ it suffices to

- map the element generators of $\text{RepoDesc}$ to elements of $A$
- map the equality generators of $\text{RepoDesc}$ to equalities between the corresponding elements of $A$
- map the equality-between-equality generators to equalities between the corresponding equalities in $A$
RepoDesc recursion

To define a function $f : \text{RepoDesc} \to A$ it suffices to give
RepoDesc recursion

To define a function \( f : \text{RepoDesc} \to A \) it suffices to give

\[
f(\text{vec}) := \ldots : A
\]
To define a function \( f : \text{RepoDesc} \rightarrow A \) it suffices to give

\[
f(\text{vec}) := \ldots : A
\]

\[
f_{1}(a\leftrightarrow b \text{ at } i) := \ldots : f(\text{vec}) = f(\text{vec})
\]
To define a function \( f : \text{RepoDesc} \rightarrow A \) it suffices to give

\[
\begin{align*}
f(\text{vec}) & := \ldots : A \\
f_1(a \leftrightarrow b \text{ at } i) & := \ldots : f(\text{vec}) = f(\text{vec}) \\
f_2(\text{compose } a \ b \ c \ d \ i \ j \ i \neq j) & := \ldots \\
& : f_1((a \leftrightarrow b \text{ at } i) o (c \leftrightarrow d \text{ at } j)) \\
& = f_1((c \leftrightarrow d \text{ at } j) o (a \leftrightarrow b \text{ at } j))
\end{align*}
\]
RepoDesc recursion

To define a function \( f : \text{RepoDesc} \rightarrow A \) it suffices to give

\[
\begin{align*}
  f(\text{vec}) & := \ldots : A \\
  f_1(a \leftrightarrow b \text{ at } i) & := \ldots : f(\text{vec}) = f(\text{vec}) \\
  f_2(\text{compose } a \ b \ c \ d \ i \ j \ i \neq j) & := \ldots \\
  & : f_1((a \leftrightarrow b \text{ at } i) \circ (c \leftrightarrow d \text{ at } j)) \\
  & = f_1((c \leftrightarrow d \text{ at } j) \circ (a \leftrightarrow b \text{ at } j))
\end{align*}
\]

You only specify \( f \) on generators, \textbf{not} \( \text{id}, \circ, !, \text{group laws}, \text{congruence}, \ldots \)

(1 patch and 4 basic axioms, instead of 4 and 14!)
RepoDesc recursion

To define a function \( f : \text{RepoDesc} \rightarrow A \) it suffices to give

\[
f(\text{vec}) := \ldots : A
\]

\[
f_1(a \leftrightarrow b \text{ at } i) := \ldots : f(\text{vec}) = f(\text{vec})
\]

\[
f_2(\text{compose } a\ b\ c\ d\ i\ j\ i \neq j) := \ldots : f_1((a \leftrightarrow b \text{ at } i) o (c \leftrightarrow d \text{ at } j)) = f_1((c \leftrightarrow d \text{ at } j) o (a \leftrightarrow b \text{ at } j))
\]
RepoDesc recursion

To define a function \( f : \text{RepoDesc} \to A \) it suffices to give

\[
\begin{align*}
f(\text{vec}) & := \ldots : A \\
f_1(a \leftrightarrow b \text{ at } i) & := \ldots : f(\text{vec}) = f(\text{vec}) \\
f_2(\text{compose } a \ b \ c \ d \ i \ j \ i \neq j) & := \ldots \\
& \quad : f_1((a \leftrightarrow b \text{ at } i) o (c \leftrightarrow d \text{ at } j)) \\
& \quad = f_1((c \leftrightarrow d \text{ at } j) o (a \leftrightarrow b \text{ at } j))
\end{align*}
\]

Type-generic equality rules say that functions act homomorphically on \( \text{id}, o, \!, \ldots \).
To define a function \( f : \text{RepoDesc} \to A \) it suffices to give

\[
\begin{align*}
  f(\text{vec}) & := \ldots : A \\
  f_1(a \leftrightarrow b \text{ at } i) & := \ldots : f(\text{vec}) = f(\text{vec}) \\
  f_2(\text{compose } a \ b \ c \ d \ i \ j \ i \neq j) & := \ldots \\
    & = f_1((a \leftrightarrow b \text{ at } i) o (c \leftrightarrow d \text{ at } j)) \\
    & = f_1((c \leftrightarrow d \text{ at } j) o (a \leftrightarrow b \text{ at } j))
\end{align*}
\]

*Type-generic equality rules say that functions act homomorphically on id, o, !, ...*
To define a function $f : \text{RepoDesc} \to A$ it suffices to give

$$f(\text{vec}) := \ldots : A$$
$$f_1(a \leftrightarrow b \text{ at } i) := \ldots : f(\text{vec}) = f(\text{vec})$$
$$f_2(\text{compose } a \ b \ c \ d \ i \ j \ i \neq j) := \ldots : f_1((a \leftrightarrow b \text{ at } i)o(c \leftrightarrow d \text{ at } j))$$
$$= f_1((c \leftrightarrow d \text{ at } j)o(a \leftrightarrow b \text{ at } j))$$
To define a function $f : \text{RepoDesc} \to A$ it suffices to give

\[
f(\text{vec}) := \ldots : A \\
f_1(\text{a} \leftrightarrow \text{b at } i) := \ldots : f(\text{vec}) = f(\text{vec})
\]

\[
f_2(\text{compose } \text{a }\text{b }\text{c }\text{d }i\ j\ i\neq j) := \ldots \\
:\quad f_1((\text{a} \leftrightarrow \text{b at } i) \circ (\text{c} \leftrightarrow \text{d at } j)) \\
= f_1((\text{c} \leftrightarrow \text{d at } j) \circ (\text{a} \leftrightarrow \text{b at } j))
\]

\text{All functions on RepoDesc respect patches}

\text{All functions on patches respect patch equality}
Patches as a HIT

1. How do you define Patch using a higher inductive type?

2. What is the elimination rule for RepoDesc?

3. How do you use the elim. rule to define interp?
Interp

Goal is to define:

\[ \text{interp} : \text{vec} = \text{vec} \]

\[ \rightarrow \text{Bijection} \ (\text{Vec Char n}) \ (\text{Vec Char n}) \]

\[ \text{interp(id)} = (\lambda x. x, \ldots) \]

\[ \text{interp}(q \circ p) = (\text{interp} \ q) \circ_b (\text{interp} \ p) \]

\[ \text{interp}(!p) = !_b (\text{interp} \ p) \]

\[ \text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat} \ a \ b \ i \]
Interp

Goal is to define:

\[ \text{interp} : \text{vec} = \text{vec} \]

\[ \rightarrow \text{Bijection} \ (\text{Vec} \ \text{Char} \ n) \ (\text{Vec} \ \text{Char} \ n) \]

\[ \text{interp}(\text{id}) = (\lambda x. x, \ldots) \]

\[ \text{interp}(q \circ p) = (\text{interp} \ q) \circ_b (\text{interp} \ p) \]

\[ \text{interp}(\neg p) = \neg_b (\text{interp} \ p) \]

\[ \text{interp}(a \leftrightarrow b \ \text{at} \ i) = \text{swapat} \ a \ b \ i \]

But only tool available is RepoDesc recursion: no direct recursion over proofs of equality
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

Need to pick A and define

\[ f(\text{vec}) := \ldots : \text{A} \]
\[ f_1(\text{a↔b at i}) := \ldots : f(\text{vec}) = f(\text{vec}) \]
\[ f_2(\text{compose}) := \ldots \]
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

Key idea: pick $A = \text{Type}$ and define

\[
f(\text{vec}) := \ldots : \text{Type}
\]
\[
f_1(a↔b \text{ at } i) := \ldots : f(\text{vec}) = f(\text{vec})
\]
\[
f_2(\text{compose}) := \ldots
\]
interp : vec = vec
   → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

Key idea: pick A = Type and define
f(vec) := Vec Char n : Type
f₁(a↔b at i) := ... : f(vec) = f(vec)
f₂(compose) := ...
interp : vec = vec
    → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

Key idea: pick A = Type and define
  f(vec) := Vec Char n : Type
  f₁(a↔b at i) := ... : Vec Char n = Vec Char n
  f₂(compose) := ...
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

Key idea: pick A = Type and define

f(vec) := Vec Char n : Type
f₁(a↔b at i) := ua(swapat a b i) : Vec Char n = Vec Char n
f₂(compose) := ...
\[
\text{interp} : \text{vec} = \text{vec} \\
\rightarrow \text{Bijection} \ (\text{Vec Char } n) \ (\text{Vec Char } n)
\]
\[
\text{interp}(a \leftrightarrow b \text{ at } i) = \text{swapat } a \ b \ i
\]

**Key idea:** pick \( A = \text{Type} \) and define

\[
f(\text{vec}) := \text{Vec Char } n : \text{Type} \\
f_1(a \leftrightarrow b \text{ at } i) := \text{ua}(\text{swapat } a \ b \ i) \quad : \text{Vec Char } n = \text{Vec Char } n
\]
\[
f_2(\text{compose}) := \ldots
\]

\text{Voevodky’s univalence axiom} \ \supset

bijective types are equal
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(a↔b at i) = swapat a b i

**Key idea: pick A = Type and define**

\[ f(\text{vec}) := \text{Vec Char } n : \text{Type} \]
\[ f_1(a↔b \text{ at } i) := ua(\text{swapat } a b i) \]
\[ : \text{Vec Char } n = \text{Vec Char } n \]
\[ f_2(\text{compose}) := \langle \text{proof about swapat as before} \rangle \]
interp : \texttt{vec = vec} \\
\rightarrow \text{Bijection} \ (\texttt{Vec Char n}) \ (\texttt{Vec Char n}) \\
interp(a \leftrightarrow b \text{ at } i) = \texttt{swapat a b i}

\textit{Key idea: pick } A = \text{Type and define}

\begin{align*}
\texttt{I}(\texttt{vec}) & := \texttt{Vec Char n : Type} \\
\texttt{I}_1(a \leftrightarrow b \text{ at } i) & := \texttt{ua(swapat a b i)} \\
\texttt{I}_2(\texttt{compose}) & := <\text{proof about swapat as before}> 
\end{align*}
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(p) = ua⁻¹(I₁(p))

Key idea: pick A = Type and define
I(vec) := Vec Char n : Type
I₁(a ↔ b at i) := ua(swapat a b i)
                 : Vec Char n = Vec Char n
I₂(compose) := <proof about swapat as before>
interp : vec = vec → Bijection (Vec Char n) (Vec Char n)
interp(p) = ua^{-1}(I_1(p))

Satisfies the desired equations (as propositional equalities):
interp(id) = (λx.x, …)
interp(q o p) = (interp q) o_b (interp p)
interp(!p) = !_b (interp p)
interp(a↔b at i) = swapat a b i
Summary
I : RepoDesc → Type interprets RepoDesc’s as Types, patches as bijections, satisfying patch equalities
Summary

- $I : \text{RepoDesc} \rightarrow \text{Type}$ interprets RepoDesc’s as Types, patches as bijections, satisfying patch equalities.

- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to $\text{id}$, $\text{circ}$, $\text{!}$, ...
Summary

- I : RepoDesc → Type interprets RepoDesc’s as Types, patches as bijections, satisfying patch equalities

- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to i,d,o,!,...

- Univalence lets you give a computational model of equality proofs (here, patches); guaranteed to satisfy laws
Summary

- $I : \text{RepoDesc} \to \text{Type}$ interprets RepoDesc’s as Types, patches as bijections, satisfying patch equalities.

- Higher inductive elim. defines functions that respect equality: you specify what happens on the generators; homomorphically extended to $\text{id}, o, !, \ldots$.

- Univalence lets you give a computational model of equality proofs (here, patches); guaranteed to satisfy laws.

- Shorter definition and code than using quotients: 1 basic patch & 4 basic axioms of equality, instead of 4 patches & 14 equations.
Where does this programming technique come from?
Homotopy type theory
Homotopy type theory

a space is a type $A$

\begin{figure}
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Diagram of a space as a type $A$.}
\end{figure}
Homotopy type theory

A space is a type $A$

Points are elements $a : A$
Homotopy type theory

A space is a type $A$

Points are elements $a : A$

Paths are proofs of equality $p : a =_A b$
Homotopy type theory

- A space is a type $A$
- Points are elements $a : A$
- Paths are proofs of equality $p : a =_A b$
- Path operations
Homotopy type theory

A space is a type $A$

Points are elements $a : A$

Paths are proofs of equality $p : a =_A b$

Path operations

$id : a = a \ (\text{refl})$
Homotopy type theory

A space is a type A

Points are elements
\( a : A \)

Paths are proofs of equality
\( p : a =_A b \)

Path operations
- id : \( a = a \) (refl)
- !p : \( b = a \) (sym)
Homotopy type theory

points are elements
\( a : A \)

paths are proofs of equality
\( p : a =_{A} b \)

a space is a type \( A \)

path operations
\begin{align*}
\text{id} & : a = a \ (\text{refl}) \\
\text{sym} & : b = a \\
\text{trans} & : a = c \\
\end{align*}
Homotopy type theory

A space is a type $A$

Points are elements $a : A$

Paths are proofs of equality $p : a =_A b$

Path operations

$id : a = a$ (refl)

$p = b a$ (sym)

$q o p : a = c$ (trans)

Homotopies

$id o p = p$

$p o p = id$

$r o (q o p) = (r o q) o p$
Homotopy type theory

A space is a type $A$

Points are elements $a : A$

Paths are proofs of equality $p : a =_A b$

Path operations

- $id : a = a$ (refl)
- $!p : b = a$ (sym)
- $q \circ p : a = c$ (trans)

Homotopies

- $id \circ p = p$
- $!p \circ p = id$
- $r \circ (q \circ p) = (r \circ q) \circ p$
Type of equalities between $\alpha$ and $-\alpha$ is inductively generated by
Equality elimination rule

Type of equalities between \( a \) and -

is inductively generated by

Fix a type \( A \) with element \( a : A \).
For a family of types \( C(y : A, \ p : a = y) \),
to give an element of

\[
C(y, p)
\]

to all \( y \) and \( p : a = y \),
suffices to give an element of

\[
C(a, id)
\]
Composition and Assoc

_o_ : a = b \rightarrow b = c \rightarrow a = c
id \circ p = p

o-assoc : (p : a=b)(q : b=c)(r : c=d)
\rightarrow p \circ (q \circ r) = (p \circ q) \circ r
o-assoc id id id id = id
Functions are functors

\[ f : A \rightarrow B \text{ has action at all levels} \]

\[ f_1 : (a_1 \ a_2 : A) \rightarrow a_1 =_A a_2 \rightarrow f(a_1) =_B f(a_2) \]

\[ f_2 : (a_1 \ a_2 : A)(p \ p' : a_1 =_A a_2) \rightarrow p =_{a_1=a_2} p' \rightarrow f_1(p) =_{f(a_1)=f(a_2)} f_1(p') \]

and so on
The Circle

Circle $S^1$ is HIT generated by

[Diagram of a circle with an arrow labeled 'loop' and a dot labeled 'base']
The Circle

Circle $S^1$ is HIT generated by

- $\text{base} : S^1$
- $\text{loop} : \text{base} = \text{base}$
The Circle

Circle $S^1$ is HIT generated by

$$\text{base} : S^1$$
$$\text{loop} : \text{base} = \text{base}$$

*Free type:* equipped with

$$\text{id}$$
$$\text{inv} : \text{loop} \circ \text{loop}^{-1} = \text{id}$$
$$\text{loop}^{-1}$$
$$\text{loop} \circ \text{loop}$$

...
The Circle

Circle recursion:
function $S^1 \to X$ determined by

base’ : X
loop’ : base’ = base’
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

id
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

id
loop
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

id
loop
loop\(^{-1}\)
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

\(\text{id}\)
\(\text{loop}\)
\(\text{loop}^{-1}\)
\(\text{loop} \circ \text{loop}\)
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

id
loop
loop⁻¹
loop o loop
loop⁻¹ o loop⁻¹
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

id
loop
loop\(^{-1}\)
loop \circ loop
loop\(^{-1}\) \circ loop\(^{-1}\)
loop \circ loop\(^{-1}\)
How many different loops are there on the circle, up to *homotopy*?

- \(\text{id}\)
- \(\text{loop}\)
- \(\text{loop}^{-1}\)
- \(\text{loop \circ \ loop}\)
- \(\text{loop}^{-1} \circ \text{loop}^{-1}\)
- \(\text{loop} \circ \text{loop}^{-1} = \text{id}\)
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

\[
\begin{align*}
\text{id} & \quad \Rightarrow & \quad 0 \\
\text{loop} & \quad \Rightarrow & \quad \text{loop} \\
\text{loop}^{-1} & \quad \Rightarrow & \quad \text{loop}^{-1} \\
\text{loop} \circ \text{loop} & \quad \Rightarrow & \quad \text{loop} \circ \text{loop}^{-1} = \text{id}
\end{align*}
\]
Fundamental group of circle

How many different loops are there on the circle, up to *homotopy*?

\[
\begin{align*}
\text{id} & \quad 0 \\
\text{loop} & \quad 1 \\
\text{loop}^{-1} & \\
\text{loop} \circ \text{loop} & \\
\text{loop}^{-1} \circ \text{loop}^{-1} & \\
\text{loop} \circ \text{loop}^{-1} & = \text{id}
\end{align*}
\]
How many different loops are there on the circle, up to homotopy?

\[
\begin{align*}
\text{id} & \quad 0 \\
\text{loop} & \quad 1 \\
\text{loop}^{-1} & \quad -1 \\
\text{loop} \circ \text{loop} & \\
\text{loop}^{-1} \circ \text{loop}^{-1} & \\
\text{loop} \circ \text{loop}^{-1} & = \text{id}
\end{align*}
\]
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

<table>
<thead>
<tr>
<th>Loop</th>
<th>Homotopy Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0</td>
</tr>
<tr>
<td>loop</td>
<td>1</td>
</tr>
<tr>
<td>loop(^{-1})</td>
<td>-1</td>
</tr>
<tr>
<td>loop (\circ ) loop</td>
<td>2</td>
</tr>
<tr>
<td>loop(^{-1}) (\circ ) loop(^{-1})</td>
<td></td>
</tr>
<tr>
<td>loop (\circ ) loop(^{-1})</td>
<td>id</td>
</tr>
</tbody>
</table>
Fundamental group of circle

How many different loops are there on the circle, up to \textit{homotopy}?

\begin{align*}
\text{id} & \quad 0 \\
\text{loop} & \quad 1 \\
\text{loop}^{-1} & \quad -1 \\
\text{loop} \circ \text{loop} & \quad 2 \\
\text{loop}^{-1} \circ \text{loop}^{-1} & \quad -2 \\
\text{loop} \circ \text{loop}^{-1} & = \text{id}
\end{align*}
Fundamental group of circle

How many different loops are there on the circle, up to homotopy?

<table>
<thead>
<tr>
<th>Loop</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>0</td>
</tr>
<tr>
<td>loop</td>
<td>1</td>
</tr>
<tr>
<td>loop⁻¹</td>
<td>-1</td>
</tr>
<tr>
<td>loop o loop</td>
<td>2</td>
</tr>
<tr>
<td>loop⁻¹ o loop⁻¹</td>
<td>-2</td>
</tr>
<tr>
<td>loop o loop⁻¹ = id</td>
<td>0</td>
</tr>
</tbody>
</table>
**Theorem.** Group of loops on the circle is isomorphic to $\mathbb{Z}$

**Proof:** Define universal cover

![Diagram of universal cover]

- Base $S^1$ mapped to $\mathbb{R}$ through the universal cover.
Theorem. Group of loops on the circle is isomorphic to \( \mathbb{Z} \)

Proof: Define universal cover

\[
\text{Cover} : S^1 \to \text{Type}
\]
\[
\text{Cover}(\text{base}) := \mathbb{Z}
\]
\[
\text{Cover}_1(\text{loop}) := \text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}
\]
**Fundamental group of circle**

**Theorem.** Group of loops on the circle is isomorphic to $\mathbb{Z}$

**Proof:** Define universal cover

\[
\text{Cover} : S^1 \rightarrow \text{Type}
\]

\[
\text{Cover}(\text{base}) := \mathbb{Z}
\]

\[
\text{Cover}_1(\text{loop}) := \text{ua}(\text{successor}) : \mathbb{Z} = \mathbb{Z}
\]

interpret loop as “add 1” bijection
Homotopy in HoTT

\[ \pi_1(S^1) = \mathbb{Z} \]
\[ \pi_{k<n}(S^n) = 0 \]
Hopf fibration
\[ \pi_2(S^2) = \mathbb{Z} \]
\[ \pi_3(S^2) = \mathbb{Z} \]
James Construction
\[ \pi_4(S^3) = \mathbb{Z}? \]

Freudenthal
\[ \pi_n(S^n) = \mathbb{Z} \]
K(G, n)
Cohomology axioms
Blakers-Massey

Van Kampen
Covering spaces
Whitehead for n-types

[Brunerie, Finster, Hou, Licata, Lumsdaine, Shulman]
What’s next?

- Operational semantics of HITs and univalence is still an open problem in general, though some special cases are known.
- Have just started exploring programming applications.
- Extensions to this example: more realistic basic patches, patches that can fail (partial bijections), implement merge.