Proof Search in Cartesian Cubical Type Theory

Zhuyang Wang Kuen-Bang Hou (Favonia)

University of Minnesota

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
```

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
```

Proof Search

- Automatically find a proof term
- Save time and effort
- Lower the barrier for beginners

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

Proof Search

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p₁ i
  | i=1 ∨ k=0 => p₂ j i
  | k=1       => p₃ j i
  ]
}
```

Automatic hcomp in Cubical Agda

Maximilian Doré, *Automating an composition* [1]

An external tool for Cubical Agda

[1] https://europroofnet.github.io/assets/wg6/stockholm-kickoff-slides/dore-europroofnet-stockholm-slides.pdf

Automatic hcomp in Cubical Agda

Maximilian Doré, *Automating an composition* [1]

An external tool for Cubical Agda

Generate `hcomp` with only paths and intervals with $\vee$ and $\wedge$.

$$
\begin{aligned}
&x, y, z, \ldots, \\
&p\ 0, p\ 1, p\ i, p\ j, p\ (i \vee j), p\ (i \wedge j), \ldots, \\
&\alpha\ 0\ 0, \alpha\ 0\ 1, \alpha\ i\ i, \alpha\ i\ j, \alpha\ (i \vee j)\ j, \alpha\ (i \vee j)\ (i \wedge j), \ldots,
\end{aligned}
$$

Then solve a constraint satisfaction problem to check the boundaries are compatible.

[1] https://europroofnet.github.io/assets/wg6/stockholm-kickoff-slides/dore-europroofnet-stockholm-slides.pdf

Automatic hcom in cooltt (Ideas)

- De Morgan $\to$ Cartesian
  - No connectives!
- Internal in cooltt
  - Reuse core language, type checker
- Heuristic for path generation
  - `let`-bindings
  - Parameters
  - User hints
- Checking constraints when picking paths

Automatic hcom?

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p₁ i
  | i=1 ∨ k=0 => p₂ j i
  | k=1       => p₃ j i
  ]
}
```

Automatic hcom?

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

``Trivial'' Proof Search

```
def funext (A : type) (B : type)
           (f : (x : A) → B) (g : (x : A) → B)
           (h : (x : A) → path B {f x} {g x})
    : path {(x : A) → B} f g :=
  i x => h x i
```

``Trivial'' Proof Search

```
def funext (A : type) (B : type)
           (f : (x : A) → B) (g : (x : A) → B)
           (h : (x : A) → path B {f x} {g x})
    : path {(x : A) → B} f g :=
  i x => h x i
```

- Hard and undecidable
- Naive search for (partial) elements
- Limited to local definitions and user hints
- Try without unification

Missing Pieces

```
def refl≡p∘p⁻¹ (A : type) (p : 𝕀 → A)
  : path {path A {p 0} {p 0}}
         {refl A {p 0}}
         {trans A p {symm A p}} :=
k i =>
hcom A 0 1 {i=0 ∨ i=1 ∨ k=0 ∨ k=1} {j =>
  [ j=0 ∨ i=0 => p i
  | i=1 ∨ k=0 => lemma₁ A p j i
  | k=1       => lemma₂ A p {lemma₃ A p} j i
  ]
}
```

Conclusion

To bring proof search to Cartesian cubical type theory in cooltt

Extensions:

- Diagonals
- Other composition operations
- Investigate different cubical type theories
- Unification