Jason Gross' Wishlist for Coq

POPL 2014 — Coq Users Meeting

Higher Inductive Types

Higher inductive types are:

Higher Inductive Types

Higher inductive types are:

- Inductive types

Higher Inductive Types

Higher inductive types are:

- Inductive types
- freely generated with higher path structure (non-trivial equalities)

Higher Inductive Types

Higher inductive types are:

- Inductive types
- freely generated with higher path structure (non-trivial equalities)

Example: The interval $(0 \rightsquigarrow 1)$

Higher Inductive Types

Higher inductive types are:

- Inductive types
- freely generated with higher path structure (non-trivial equalities)

Example: The interval $(0 \rightsquigarrow 1)$

```
Inductive Interval :=
| zero : Interval
| one  : Interval
| seg  : zero = one.
```

Higher Inductive Types

Why?

Higher inductive types are useful for:

Higher Inductive Types

Why?

Higher inductive types are useful for:

- Homotopy type theory (making basic spaces)

Higher Inductive Types

Why?

Higher inductive types are useful for:

- Homotopy type theory (making basic spaces)
- Quotient types

Higher Inductive Types

Why?

Higher inductive types are useful for:

- Homotopy type theory (making basic spaces)
- Quotient types
- Formalizing version control systems (according to Dan Licata[1])
- Proving functional extensionality

[1] "Git as a HIT",
http://dlicata.web.wesleyan.edu/pubs/l13git/git.pdf

Higher Inductive Types

Proving functional extensionality

```
Definition functional_extensionality A B f g
    (H : ∀ x, f x = g x ) → f = g
  := f_equal
      (λ i x ⇒
        match i return B with
          | zero ⇒ f x
          | one  ⇒ g x
          | seg  ⇒ H x
        end)
      seg.
```

Higher Inductive Types

Proving functional extensionality

```
:= match seg in (_ = y)
     return ((λ x ⇒ f x)
             = (λ x ⇒ match y with
                       | zero ⇒ f x
                       | one  ⇒ g x
                       | seg  ⇒ H x
                       end))
   with
     | eq_refl => eq_refl
   end.
```

Higher Inductive Types

How?

Note that higher inductive types don't magically give you computational functional extensionality.

Higher Inductive Types

How?

Note that higher inductive types don't magically give you computational functional extensionality.

You must solve computational functional extensionality to implement computational higher inductive types.

The following features, which I want, are slated for Coq 8.5: universe polymorphism parallel processing tactics in terms faster sigma types and projections eta for records

The following are features that I want: a better story for namespacing (https://coq.inria.fr/bugs/show_bug.cgi?id=3171) irrelevant fields/a type of strict hProps/Coq* built-in CoqMT judgmental eta for the unit type a search that searches the entire standard library, and not just currently [Require]d files a search which is up to unification, rather than up to pattern matching coercions which don't care about the uniform inheritance condition (see also https://coq.inria.fr/bugs/show_bug.cgi?id=3115) built-in MTac or other monadic tactic language better handling of open terms in ltac, and support for recursing under binders in tactics (maybe fixed with new tactic engine? see also https://coq.inria.fr/bugs/show_bug.cgi?id=3106 and https://coq.inria.fr/bugs/show_bug.cgi?id=3102) easier use of ML plugins (I don't want to have to recompile them myself)