Computational Higher Inductive Types
Computing with Custom Equalities

Jason Gross
jgross@mit.edu

MIT CSAIL Student Workshop

April 10, 2014
Properties of Equality

Warm Up: Linked Lists

Example: Unordered Sets
   Canonical Inhabitants
   Higher Inductive Types

Computing with Higher Inductive Types

Thank you
Properties of Equality

- Reflexivity: \( x = x \)
- Symmetry: if \( x = y \) then \( y = x \)
- Transitivity: if \( x = y \) and \( y = z \), then \( x = z \)
- Leibniz rule: if \( x = y \), then \( f(x) = f(y) \)
Warm Up: Linked Lists

- Two constructors: nil, or [], and cons
- Two accessors on non-nil lists: head and tail
- Equality is defined on an element-by-element basis
  - [] = []
  - [] ≠ [a, ...]
  - [a, ...] ≠ []
  - [x_0, x_1, ..., x_n] = [y_0, y_1, ..., y_m] iff [x_1, ..., x_n] = [y_1, ..., y_m]
    and x_0 = y_0
- Fairly easy to prove the properties of equality
  - In Coq, Agda, and Idris, you get all of these properties for free
Example: Unordered Sets

- `nil`, or `∅`
- `add`
- `remove`
- `contains`
- Often implemented internally as a list or a tree
- Equality is then implemented as “is one a permutation of the other?"
- Fairly easy to prove that it’s an equivalence relation
- Leibniz rule (if $x = y$, then $f(x) = f(y)$) is harder
- In Haskell, Agda, Coq, and Idris, the Leibniz rule is false! (or at least not internally provable)
  - The problem is that either you don’t have private fields, or you can’t make use of the fact that everything is defined in terms of your public methods.
Example: Unordered Sets

Solution 1: Canonical Inhabitants

- Give up private fields, but use element-wise equality
- Define a type of “sorted lists without duplication”, and call them sets
- Now we can use element-wise equality, and get Leibniz (and other properties) for free
- What if we don’t have an ordering on the elements, only equality?
- Is this really what we wanted? We asked for unordered sets, and instead made sorted lists.
Example: Unordered Sets
Solution 2: Higher Inductive Types

- Higher Inductive Types
- Keep the built-in equality (so we get the properties for free), but turn it into equality up to permutation
- How do we get that it’s an equivalence relation for free?
  - Take the reflexive symmetric transitive closure of the given relation
- How do we get Leibniz for free?
  - Require proving it each time you define a particular function
  - To define a function that deals with unordered sets, you have to simultaneously prove that your function is invariant under permutations
It seems simple enough, so what’s the problem?

Having higher inductive types gives you functional extensionality (if \( f(x) = g(x) \) for all \( x \), then \( f = g \)), which doesn’t yet have a good computational interpretation in Coq nor Agda nor Idris.

Equality in Coq and Agda (--without-K) actually has a rich structure.

If you look at proofs of equality, and equality of these proofs, and you iterate this process, you get enough math to do topology!

This is Homotopy Type Theory.
Thank you

Thanks!

Questions?
Example: Unordered Sets

Solution 3: Parametricity

- Make use of the fact that private fields are private
- Very hard to do!
- Can probably be done by way of parametricity (aka “theorems for free”), or a generalization of it
- Parametricity can be given a computational interpretation, but it’s very non-trivial to do so