Private inductive types

July 2013
Introduction

- Higher Inductive types: adding equalities
- Preventing inconsistencies
- Preserving convertibility
- Simulating with private types
What is this thing called Equality

- A family of equality types: for $x, y : A$, $x = y$ is a type
- Described as an inductive type: no specific treatment
- Induction principle illuminating
  \[
  \forall A : Type. \forall x : A. \\
  \forall P : A \rightarrow Prop. P(x) \Rightarrow \forall y : A. x = y \Rightarrow P(y)
  \]
- If $x = y$ then every property satisfied by $x$ is also satisfied by $y$
- $x$ and $y$ are undistinguishable
  - Are they really?
Say that when $x = y$, then $x$ and $y$ are not really the same for all purposes.

So $x = y$ should only mean *there is a path between $x$ and $y*.

Distinction at a microscopic level.

But at the macroscopic level, still $x$ and $y$ are equal.
Build new objects with paths between them

- State at the same time the creation of objects and the property that they are identical.
- Example: assert the existence of two points N and S and two paths between them.
- Already done easily for points using inductive types
- What about the paths?
  - Natural to add paths as axioms
Inconsistencies with axiomatic paths

- Usual interpretation of equality (identity) types
  - Ultimately only one way to build proofs of equality: reflexivity
- No confusion property of inductive types
  - Rely on strong elimination
- Axiomatic paths between constructors incompatible with no-confusion
Illustration

Inductive cellc :=
  N | S.

Axiom west : N = S.
Axiom east : N = S.

- Obviously inconsistent in plain Coq.
Preventing inconsistency

- Allow only to define function that preserve path consistency
- In illustration, \( f \) \( \mathbb{N} \) and \( f \) \( \mathbb{S} \) must have a path between them.
- Also take into account dependent types
- Solution already easy to implement in Agda
Heavy solution

- Avoid inductive types
- State axioms for all elements of the higher inductive type
Illustrating the heavy solution

Parameters (cellc : Type) (N S : cellc).
Axioms west east : N = S.

Parameter cellc_rect (P : cellc -> Type)
  (vn : P N) (vs : P S)
  (pw : eq_rect N P vn S west = vs)
  (pe : eq_rect N P vn S east = vs)
  (x : cellc) : P x.

Axiom cellc_rect_N :=
  forall P vn vs pw pe, cellc_rect P vn vs pw pe N = vn.

Axiom cellc_rect_S :=
  forall P vn vs pw pe, cellc_rect P vn vs pw pe S = vs.
What’s wrong with being heavy?

- *Provably equal* is not *convertible*
  - `cellc_rect P vn vs pw pe N` and `vn` are not convertible
- More uses of `eq_rect` are required everywhere
- The size of proofs increases drastically
Adding convertibility

- Come back to inductive types
- Design elimination function to enforce guarantees

Definition cellc_rect (P : cellc → Type)
  (vn : P N) (vs : P S)
  (pw : eq_rect N P vn S west = vs)
  (pe : eq_rect N P vn S east = vs) (x : cellc) :=
match x return P x with N => vn | S => vs end.
Computing with cellc_rect

- `cellc_rect P vn vs pw pe N` and `vn` are now convertible.
- Okay if the only functions definable in Coq have to be defined using `cellc_rect`.
- Need to forbid direct use of pattern-matching, tactics `case`, `discriminate`, `inversion`, `injection`...
Idea of private types

- In a module, define an inductive type to be private
- Inside module: unsafe operations, trusting the programmer
- Outside module: more safety, only functions provided by module designer
- Preserve computation (convertibility) for functions provided in the module
- No modification of the kernel, only module handling
- Deactivate tactics and syntax
- Hard questions about consistency: not treated by the kernel
Simulating the circle inductive type

Module Circle.
Local Inductive Circle := N | S.
Axiom east : N = S.
Axiom west : N = S.

Definition circle_induction (A : Type)(vn : A)(vs : A)
  (epd : vn = vs)(wpd : vn = vs)(x : circle) : A :=
  match x with N => vn | S => vs end.

Axiom circle_induction_cws :
  forall A vn vs epd wpd,
    ap (circle_induction vn vs epd wpd) east_side = epd.

End Circle.
Conclusion

- Potential inconsistency comes from adding axioms
- Idea of private types orthogonal to axioms
- Application outside homotopy theory are probable