Dependent Type Theory

Powerful, expressive theory, esp. for formalisation of maths.
(Precisely: Martin-Löf TT, CIC, etc.; implemented in Coq, Agda, etc.)

Key judgements: terms, types in contexts.

$$x_1 : A_1,\ \ldots\ x_n : A_n \vdash B \ \ \mathsf{type}$$

$$x_1 : A_1,\ \ldots\ x_n : A_n \vdash b : B$$

Internal predicate logic: via Curry-Howard.

Propositional Equality

Predicate representing equality: "identity types".

$$\frac{}{x, y : A \vdash \mathrm{Id}_A(x, y)\ \mathsf{type}} \qquad \frac{}{x : A \vdash r(x) : \mathrm{Id}_A(x, x)}$$

$$\frac{\begin{array}{c} x, y : A,\ u : \mathrm{Id}_A(x, y) \vdash C(x, y, u)\ \mathsf{type} \\ z : A \vdash d : C(z, z, r(z)) \end{array}}{x, y : A,\ u : \mathrm{Id}_A(x, y) \vdash J_C(z.d; x, y, u) : C(x, y, u)}$$

Exercise. "All constructions invariant under equality", aka "transport between fibers":

$$\frac{x : A \vdash B(x)\ \mathsf{type}}{x, x' : A,\ u : \mathrm{Id}_A(x, x'),\ y : B(x) \vdash \mathsf{transport}_B(y, u) : B(x')}$$

Problem?

Id-types: simple rules, proof-theoretically lovely.

But: unintuitive behaviour. Equality on a set, surely: just a proposition — no computational content??

$\mathrm{Id}(x, y)$ can be a non-trivial type, containing distinct terms.

Solution: Extensional Type Theory?

Can impose axioms to trivialise identity types.

Force types to conform to our intuition about sets.

Unfortunately: destroys computational content, decidability of type-checking, …

Solution: "Homotopy Type Theory"

Or: change our idea of what types look like

More like *spaces* or *(higher) categories*.

Propositional equalities: *paths* in a space, *morphisms* in a category...

(Precise results: eg models of DTT in **Top**; syntax forms ω-groupoids; ...)

Payoffs

Various advantages:

- Logic to natively formalise homotopy theory.
- Intuition for working with Id-types.
- "Univalence Axiom": equality between types can be isomorphism. Gives (some) parametricity for free: all constructions on types invariant under iso.

Higher Inductive Types

Why not… axiomatise ~~topological~~ spaces directly?

```
Inductive Circle : Type where
  | base : Circle
  | loop : Paths base base.
```

“Look Ma, no reals!”

(Relatively) quick route to formalising (some?) homotopy theory.

Thank you!

References, related reading, Coq files, and much more at:
http://homotopytypetheory.org