To Infinity, and Beyond: From Setoids to Weak ω-Categories

Thanks to Nicolai Krauss, Dan Licata, Darin Morrison and Ondrej Rypacek

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

October 8, 2011

Theorem proving in Agda

$$
\begin{aligned}
&\_+\_ : \mathbb{N} \longrightarrow \mathbb{N} \longrightarrow \mathbb{N} \\
&\mathit{zero} \quad + n = n \\
&\mathit{suc}\ m + n = \mathit{suc}\ (m + n)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{assoc} : \{i\ j\ k : \mathbb{N}\} \longrightarrow i + (j + k) \equiv (i + j) + k \\
&\mathit{assoc}\ \mathit{zero} \quad j\ k = \mathit{refl} \\
&\mathit{assoc}\ (\mathit{suc}\ i)\ j\ k = \mathit{cong}\ \mathit{suc}\ (\mathit{assoc}\ i\ j\ k)
\end{aligned}
$$

- Exploit Curry-Howard.
- Think of proofs as programs.
- Termination checker to achieve logical soundness.

Basic ingredients of Type Theory

Per Martin-Löf

Π-types $(x : A) \longrightarrow B\ x$ or $\{x : A\} \longrightarrow B\ x$

- Generalize function types ($A \longrightarrow B$).
- Represent universal quantification
- Alternative syntax: $\Pi\ [x : A]\ B\ x$

Σ-types $\Sigma\ [x : A]\ B\ x$

- Generalize product types
- Represent existential quantification
- Usually curried away or replaced by datatypes

Equality types $a \equiv b$ (for $a\ b : A$)

- No simply typed correspondence
- Represent propositional equality
- Implicitly used in dependent datatypes (like *Vec* or *Fin*)

Equality types

- Equality types in Type Theory: $a \equiv b$ is the set of proofs that a is equal to b.

$$\begin{array}{l} \textbf{data}\ \_\equiv\_ : A \longrightarrow A \longrightarrow \mathit{Set}\ \textbf{where} \\ \quad \mathit{refl} : \{a : A\} \longrightarrow a \equiv a \end{array}$$

- We can show that $\equiv$ is an equivalence relation using pattern matching.

$$\begin{array}{l} \mathit{sym} : a \equiv b \longrightarrow b \equiv a \\ \mathit{sym}\ \mathit{refl} = \mathit{refl} \end{array}$$

$$\begin{array}{l} \mathit{trans} : a \equiv b \longrightarrow b \equiv c \longrightarrow a \equiv c \\ \mathit{trans}\ \mathit{refl}\ q = q \end{array}$$

About equality proofs

- In Type Theory we can make statements about the equality of equality proofs.
- E.g. *Uniqueness of Identity Proofs* (UIP) : all equality proofs are equal.

$$uip : (p\ q : a \equiv b) \longrightarrow p \equiv q$$

- We may ask wether equality is a groupoid, i.e.

$$\begin{aligned}
lneutr &: trans\ refl\ p \equiv p \\
rneutr &: trans\ p\ refl \equiv p \\
assoc &: trans\ (trans\ p\ q)\ r \equiv trans\ p\ (trans\ q\ r) \\
linv &: trans\ (sym\ p)\ p \equiv refl \\
rinv &: trans\ p\ (sym\ p) \equiv refl
\end{aligned}$$

Pattern matching proves UIP

- All the equalities are provable using pattern matching, e.g.

$$\begin{array}{l} \mathit{uip} : (p\ q : a \equiv b) \longrightarrow p \equiv q \\ \mathit{uip}\ \mathit{refl}\ \mathit{refl} = \mathit{refl} \end{array}$$

J - the eliminator

- An alternative to pattern matching is the eliminator J:

$$
\begin{aligned}
&J : (M : \{a\, b : A\} \longrightarrow a \equiv b \longrightarrow \mathit{Set}) \\
&\quad \longrightarrow (\{a : A\} \longrightarrow M\ (\mathit{refl}\ \{a\})) \\
&\quad \longrightarrow (p : a \equiv b) \longrightarrow M\ p \\
&J\ M\ m\ (\mathit{refl}\ \{a\}) = m\ \{a\}
\end{aligned}
$$

- Using J we can derive all the previous propositions but not *uip*.
- J corresponds to a restricted form of pattern matching.

Question

Should we accept or reject UIP?

Equality of functions

- What should be equality of functions?
- All operations in Type Theory preserve extensional equality of functions.
  The only exception is intensional propositional equality.
- We would like to define propositional equality as extensional equality.

$$
\begin{array}{l}
\textit{postulate} \\
\quad \textit{ext} : (f\ g : A \longrightarrow B) \\
\qquad \longrightarrow ((a : A) \longrightarrow f\ a \equiv g\ a) \longrightarrow f \equiv g
\end{array}
$$

Equality of types

- What should be equality of types?
- All operations of Type Theory preserve isomorphisms (or bijections).
  The only exception is intensional propositional equality.
- Unlike Set Theory, e.g. $\{0,1\} \simeq \{1,2\}$ but $\{0,1\} \cup \{0,1\} \not\simeq \{0,1\} \cup \{1,2\}$.
- We would like to define propositional equality of types as isomorphism.

UIP and isomorphism

- UIP doesn't hold if we define equality of types as isomorphism.
- E.g. there is more than one way to prove that *Bool* is isomorphic to *Bool*.
- If we want to use isomorphism as equality we cannot allow uip.

Eliminating extensionality

- Adding principles like *ext* or univalence as constants destroys basic computational properties of Type Theory.
- E.g. there are natural numbers not reducible to a numeral.
- We can eliminate *ext* by translating every type as a setoid see my LICS 99 paper: *Extensional Equality in Intensional Type Theory*.

Setoids

- Setoids are sets with an equivalence relation.

$$
\begin{array}{l}
\mathit{record}\ \mathit{Setoid} : \mathit{Set}_1\ \textbf{where} \\
\quad \mathit{field} \\
\qquad \mathit{set} : \mathit{Set} \\
\qquad \mathit{eq} : \mathit{set} \longrightarrow \mathit{set} \longrightarrow \mathit{Prop} \\
\qquad \ldots
\end{array}
$$

- I write *Prop* to indicate that all proofs should be identified.
- This seems necessary for the construction.

Function setoids

- A function between setoids has to respect the equivalence relation.

$$
\begin{array}{l}
\textit{record}\ \_\Rightarrow\textit{set}\_\ (A\ B : \textit{Setoid}) : \textit{Set}\ \textbf{where} \\
\quad \textit{field} \\
\quad\quad \textit{app} : \textit{set}\ A \longrightarrow \textit{set}\ B \\
\quad\quad \textit{resp} : \forall \{a\}\ \{a'\} \longrightarrow \textit{eq}\ A\ a\ a' \longrightarrow \textit{eq}\ B\ (\textit{app}\ a)\ (\textit{app}\ a')
\end{array}
$$

- Equality between functions is extensional equality:

$$
\begin{array}{l}
\_\Rightarrow\_ : \textit{Setoid} \longrightarrow \textit{Setoid} \longrightarrow \textit{Setoid} \\
A \Rightarrow B = \textit{record}\ \{ \\
\quad \textit{set} = A \Rightarrow \textit{set}\ B; \\
\quad \textit{eq} = \lambda\ f\ f' \longrightarrow \\
\quad\quad \forall \{a\} \longrightarrow \textit{eq}\ B\ (\textit{app}\ f\ a)\ (\textit{app}\ f'\ a) \}
\end{array}
$$

Proof-Irrelevance

- Since we are using *Prop* the construction enforces UIP.

Question

What do we have to use instead of setoids, if we don't want UIP?

Globular sets

- The first approximation are *globular sets* which are a coinductive type:

$$
\begin{array}{l}
\textit{record Glob} : \textit{Set}_1\ \textbf{where} \\
\quad \textit{field} \\
\quad\quad \textit{obj} : \textit{Set} \\
\quad\quad \textit{eq} : \textit{obj} \longrightarrow \textit{obj} \longrightarrow \infty \textit{Glob}
\end{array}
$$

Function globular sets

- The set of functions is also defined coinductively:

$$\begin{array}{l} record\ \_\Rightarrow set\_\ (A\ B : Glob) : Set\ \textbf{where}\ field \\ \quad app : set\ A \longrightarrow set\ B \\ \quad resp : \forall\{\,a\ a'\,\} \longrightarrow \infty(\flat(eq\ A\ a\ a') \\ \qquad \Rightarrow set\ (\flat(eq\ B\ (app\ a)\ (app\ a')))) \end{array}$$

- To define equality we need Π-types as a globular set:

$$\begin{array}{l} \Pi : (A : Set)\ (F : A \longrightarrow Glob) \longrightarrow Glob \\ \Pi\ A\ F = record\ \{ \\ \quad set = (a : A) \longrightarrow set\ (F\ a); \\ \quad eq = \lambda\ f\ g \longrightarrow \sharp\Pi\ A\ (\lambda\ a \longrightarrow \flat(eq\ (F\ a)\ (f\ a)\ (g\ a)))\} \end{array}$$

- Now we can define function globular sets:

$$\begin{array}{l} \_\Rightarrow\_ : Glob \longrightarrow Glob \longrightarrow Glob \\ A \Rightarrow B = record\ \{ \\ \quad set = A \Rightarrow set\ B; \\ \quad eq = \lambda\ f\ g \longrightarrow \sharp\Pi\ (set\ A)\ (\lambda\ a \longrightarrow \flat(eq\ B\ (app\ f\ a)\ (app\ g\ a)) \end{array}$$

What about the . . . ?

- For setoids we have to add:

$$
\begin{array}{l}
\mathit{record}\ \mathit{Setoid} : \mathit{Set}_1\ \textbf{where} \\
\quad \mathit{field} \\
\quad\quad \mathit{set} : \mathit{Set} \\
\quad\quad \mathit{eq} : \mathit{set} \longrightarrow \mathit{set} \longrightarrow \mathit{Prop} \\
\quad\quad \mathit{refl} : \forall \{a\} \longrightarrow \mathit{eq}\ a\ a \\
\quad\quad \mathit{sym} : \forall \{a\}\ \{b\} \longrightarrow \mathit{eq}\ a\ b \longrightarrow \mathit{eq}\ b\ a \\
\quad\quad \mathit{trans} : \forall \{a\}\ \{b\}\ \{c\} \longrightarrow \mathit{eq}\ a\ b \longrightarrow \mathit{eq}\ b\ c \longrightarrow \mathit{eq}\ a\ c
\end{array}
$$

- What do we need for globular sets?

Weak ω-groupoids

- We need *refl*, *sym* and *trans* at all levels.
- We require the groupoid equations everywhere.
- *trans* and *sym* are actually functors.
- All equalities are weak, i.e. equations are witnessed by elements of homsets.
- Coherence: All equations which are provable using a strict equality should be witnessed in the weak sense.

Globular sets

- A weak ω-groupoids is a globular set with additional structure.
- To define this framework we introduce a language to talk about categories and objects in a weak ω-groupoid.
- A weak ω-gropoid is then defined as a globular set which interprets this language.

Syntax for globular sets

$$\textbf{data}\ \ \mathrm{Con} : \mathrm{Set}\ \ \textbf{where}\ \frac{}{\varepsilon : \mathrm{Con}}\ ;\ \frac{\Gamma : \mathrm{Con} \qquad C : \mathrm{Cat}\ \Gamma}{(\Gamma, C) : \mathrm{Con}}$$

$$\textbf{data}\ \frac{\Gamma : \mathrm{Con}}{\mathrm{Cat}\ \Gamma : \mathrm{Set}}\ \textbf{where}\ \frac{}{\bullet : \mathrm{Cat}\ \Gamma}\ ;\ \frac{C : \mathrm{Cat}\ \Gamma \qquad a\ b : \mathrm{Obj}\ C}{C[\,a\,,\,b\,] : \mathrm{Cat}\ \Gamma}$$

$$\textbf{data}\ \frac{C : \mathrm{Cat}\ \Gamma}{\mathrm{Obj}\ C : \mathrm{Set}}\ \textbf{where} \frac{v : \mathrm{Var}\ C}{\mathrm{var}\ v : \mathrm{Obj}\ C}\ \cdots$$

Interpretation

A *weak* ω *category* is given by the following data:

1. A globular set G : Glob
2. $$\frac{o : \text{Obj } C \qquad x : [\![\Gamma]\!]}{[\![o]\!]\ x : \text{obj } ([\![C]\!]\ x)}$$
   $$\frac{\Gamma : \text{Con}}{[\![\Gamma]\!] : \text{Set}} \qquad \frac{}{[\![\varepsilon]\!] = 1} \qquad \frac{}{[\![\Gamma, C]\!] = \Sigma(x : [\![\Gamma]\!])([\![C]\!]\ x)}$$
   $$\frac{C : \text{Cat } \Gamma \qquad x : [\![\Gamma]\!]}{[\![C]\!]\ x : \text{Glob}}$$
   $$\frac{}{[\![\bullet]\!]\ x = G} \qquad \frac{}{[\![C[a,b]]\!]\ x = \text{hom } ([\![C]\!] x)\ ([\![a]\!]\ x)\ ([\![b]\!]\ x)}$$
3. Conditions on the interpretation of variables ...

Composability

$$\textbf{data}\ \frac{C\ D : \mathrm{Cat}\ \Gamma}{C \between D : \mathrm{Set}}\ \textbf{where}\ \frac{}{\mathrm{zero} : C[a,b] \between C[b,c]}\ ;$$

$$\frac{H : C \between D}{\mathrm{suc}\, H : C[a,b] \between D[a',b']}$$

Composition

$$\frac{C\ D : \mathsf{Cat}\ \Gamma \qquad n : C \between D}{C \circ_n D : \mathsf{Cat}\ \Gamma} \qquad \frac{a : \mathsf{Obj}\ C \qquad b : \mathsf{Obj}\ D}{a \circ_n b : \mathsf{Obj}\ (C \circ_n D)}$$

$$\frac{C\ D : \mathsf{Cat}\ \Gamma \qquad n : C \between D}{C \circ_n D : \mathsf{Cat}\ \Gamma} \qquad \frac{C : \mathsf{Cat}\ \Gamma \qquad a\ b\ c : \mathsf{Obj}\ C}{C[a,b] \circ_0 C[b,c] = C[a,c]}$$

$$\frac{n : C \between D \qquad a\ b : \mathsf{Obj}\ C \qquad c\ d : \mathsf{Obj}\ D}{C[a,b]\ \circ_{n+1}\ D[c,d]\ =\ (C \circ_n D)[a \circ_n c, b \circ_n d]}$$

Strict equality

$$\textbf{data}\ \frac{C\ D : \mathrm{Cat}\ \Gamma}{C \doteq D : \mathrm{Set}}$$

$$\frac{}{\bullet_{\doteq}\ :\ \bullet \doteq \bullet} \qquad \frac{H : C \doteq D \qquad A : H \vdash a \doteq c \qquad B : H \vdash b \doteq d}{H[A, B]_{\doteq} : H[a, b] \doteq H[c, d]}$$

$$\textbf{data}\ \frac{H : C \doteq D \qquad a : \mathrm{Obj}\ C \qquad b : \mathrm{Obj}\ D}{H \vdash a \doteq b : \mathrm{Set}}\ \textbf{where}\ \cdots$$

$$\frac{\{T : \mathsf{Tele}\ (C[a,b])\} \qquad \alpha : \mathsf{Obj}\ (T \Downarrow)}{\lambda_{\doteq}\ \alpha : \_ \vdash \alpha \circ\_\ (\mathsf{id}^{(\mathsf{depth}\ t)}\ b) \doteq \alpha}$$

$$\frac{\{T : \mathsf{Tele}\ (C[a,b])\} \qquad \alpha : \mathsf{Obj}\ (T \Downarrow)}{\rho_{\doteq}\ \alpha : \_ \vdash (\mathsf{id}^{(\mathsf{depth}\ t)}\ a) \circ\_\ \alpha \doteq \alpha}$$

$$\frac{\begin{array}{ccc}\{t : \mathsf{Tele}\ (C[a,b])\} & \{u : \mathsf{Tele}\ (C[b,c])\} & \{v : \mathsf{Tele}\ (C[b,c])\} \\ \alpha : \mathsf{Obj}\ (T \Downarrow) & \beta : \mathsf{Obj}\ (u \Downarrow) & \gamma : \mathsf{Obj}\ (v \Downarrow)\end{array}}{\alpha_{\doteq} : \_ \vdash (\alpha \circ\_\ \beta) \circ\_\ \gamma \doteq \alpha \circ\_\ (\beta \circ\_\ \gamma)}$$

$$\frac{a : \mathrm{Obj}\ C}{\mathrm{id}\ a : \mathrm{id}\ C \vdash a \doteq a} \qquad \frac{p : H \vdash a \doteq b}{p^{-1} : H^{-1} \vdash b \doteq a}$$

$$\frac{p : H \vdash a \doteq b \qquad q : I \vdash b \doteq c}{p;q : H;I \vdash a \doteq c}$$

Conclusions

- Weak ω-groupoids replace setoids when we want to interpret Type Theory without UIP.
  (*higher dimensional Type Theory*)
- Already defining them precisely is quite hard.
- Using them to interpret Type Theory looks even harder.
- Are there ways to reduce bureaucracy?