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

July 7, 2011

Equality types

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

$$\textbf{data}\ \_\equiv\_ : A \to A \to \mathit{Set}\ \textbf{where}$$
$$\mathit{refl} : \{a : A\} \to a \equiv a$$

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

$$\mathit{sym} : a \equiv b \to b \equiv a$$
$$\mathit{sym}\ \mathit{refl} = \mathit{refl}$$
$$\mathit{trans} : a \equiv b \to b \equiv c \to a \equiv c$$
$$\mathit{trans}\ \mathit{refl}\ q = q$$

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.

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

- 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.

$$\begin{aligned}&\mathit{uip} : (p\ q : a \equiv b) \to p \equiv q\\&\mathit{uip}\ \mathit{refl}\ \mathit{refl} = \mathit{refl}\end{aligned}$$

J - the eliminator

- An alternative to pattern matching is the eliminator J:

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

$$J\;M\;m\;(\mathit{refl}\;\{a\}) = m\;\{a\}$$

- 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.

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

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.

  record Setoid : Set_1 **where**
  field
  set : *Set*
  eq : *set* $\rightarrow$ *set* $\rightarrow$ *Prop*
  ...

- 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.

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

- Equality between functions is extensional equality:

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

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:

$$
\begin{array}{l}
\mathit{record}\ \mathit{Glob} : \mathit{Set}_1\ \textbf{where} \\
\quad \mathit{field} \\
\qquad \mathit{obj} : \mathit{Set}_0 \\
\qquad \mathit{eq} : \mathit{obj} \to \mathit{obj} \to \infty \mathit{Glob}
\end{array}
$$

Function globular sets

- The set of functions is also defined coinductively:

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

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

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

- Now we can define function globular sets:

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

What about the . . . ?

- For setoids we have to add:

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

- 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.

The framework

data *Con* : *Set* **where**
  ϵ : *Con*
  $\_,\_$: $(\Gamma : Con)\ (C : Cat\ \Gamma) \to Con$
record HomSpec $(\Gamma : Con) : Set$ **where**
  field
    cat : $Cat\ \Gamma$
    dom cod : $Obj\ cat$
data *Cat* : $(\Gamma : Con) \to Set$ **where**
  ffl : $\forall\ \{\Gamma\} \to Cat\ \Gamma$
  hom : $\forall \{\Gamma\} \to HomSpec\ \Gamma \to Cat\ \Gamma$
data *Obj* : $\{\Gamma : Con\}\ (C : Cat\ \Gamma) \to Set$ **where**
  var : $\forall \{\Gamma\}\ \{C : Cat\ \Gamma\} \to Var\ C \to Obj\ C$
  ...

$record\ \omega Cat : Set_1$ **where**

$\quad field$

$\qquad G : Glob$

$\qquad evalCon : Con \to Set$

$\qquad evalCat : (C : Cat\ \Gamma)\ (\gamma : evalCon\ \Gamma) \to Glob$

$\qquad evalObj : (A : Obj\ C)\ (\gamma : evalCon\ \Gamma) \to Glob.obj\ (evalCat\ C\ \gamma)$

$\qquad evalCon\ \epsilon\ G = \top$

$\qquad evalCon\ (\Gamma, C)\ G =$

$\qquad\qquad\qquad \Sigma\ [\gamma : evalCon\ \Gamma\ G]\ Glob.obj\ (evalCat\ C\ G\ \gamma)$

$\qquad evalCat$ ffl $G\ \gamma = G$

$\qquad evalCat\ (hom\ (C\ [A, B]))\ G\ \gamma = \flat(Glob.hom\ (evalCat\ C\ G\ \gamma)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (evalObj\ A\ G\ \gamma)$

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (evalObj\ B\ G\ \gamma))$

$\qquad\quad \ldots$

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?