Weak $\omega$-Groupoids in Type Theory

Based on joint work with Ondrej Rypacek

Thorsten Altenkirch

Functional Programming Laboratory
School of Computer Science
University of Nottingham

April 2, 2012
Question
Can we understand a concept by describing it categorically?

- Category theory helps us to structure and relate concepts.
- True understanding must come form somewhere else.
- Set theory?
- There is something better!
Type Theory

- Propositions are types
- Basic constructions on types: functions, tuples, enumerations, ...
- Implementations of Type Theory: Coq, Agda, ...
- Informal understanding of Type Theory!
Basic ingredients of Type Theory

**Π-types**  dependent function types
functions, implication, universal quantification

**Σ-types**  dependent pair types
tuples, conjunction, existential quantification

Finite types  0, 1, 2

Equality types  Given $a, b : A$, $a \equiv b$ is the type of proofs that $a$ is equal to $b$.

Inductive and coinductive types  Finite and infinite trees.

Universes  $\text{Set}_0 : \text{Set}_1 : \ldots$
Example: Axiom of choice

\[\text{ac} : ((a : A) \rightarrow \Sigma \ [b : B] \ R \ a \ b) \rightarrow \Sigma \ [f : (A \rightarrow B)] \ ((a : A) \rightarrow R \ a \ (f \ a))\]

\[\text{ac} \ g = (\lambda \ a \rightarrow \text{proj}_1 \ (g \ a)), (\lambda \ a \rightarrow \text{proj}_2 \ (g \ a))\]

- Follows from the constructive explanation of connectives.
- \text{ac} is actually an isomorphism, i.e. there is an inverse:

\[\text{ac}' : \Sigma \ [f : (A \rightarrow B)] \ ((a : A) \rightarrow R \ a \ (f \ a)) \rightarrow ((a : A) \rightarrow \Sigma \ [b : B] \ R \ a \ b)\]

\[\text{ac}' \ (f, g) = \lambda \ a \rightarrow f \ a, g \ a\]
Propositional equality

\[
\text{data } \_ \equiv \_ : A \rightarrow A \rightarrow \text{Set where}
\]
\[
\text{refl} : \quad a \equiv a
\]

Using pattern matching we can show that \( \_ \equiv \_ \) is an equivalence relation:

\[
\_^{-1} : a \equiv b \rightarrow b \equiv a
\]
\[
\text{refl}^{-1} = \text{refl}
\]
\[
\_ \circ \_ : b \equiv c \rightarrow a \equiv b \rightarrow a \equiv c
\]
\[
\text{refl} \circ q = q
\]
The eliminator \( J \)

Instead of pattern matching we can use the eliminator:

\[
J : (P : \{ a b : A \} \rightarrow a \equiv b \rightarrow \text{Set})
\rightarrow (\{ a : A \} \rightarrow P \{ a \} \text{ refl})
\rightarrow \{ a b : A \} \rightarrow (p : a \equiv b) \rightarrow P p
\]

\[
J P m \text{ refl } = m
\]

\[
_\leftarrow^{-1} : a \equiv b \rightarrow b \equiv a
\]

\[
_\leftarrow^{-1} = J (\lambda \{ a \} \{ b \} _\leftarrow \rightarrow b \equiv a) \text{ refl}
\]

\[
_\circ_\leftarrow : b \equiv c \rightarrow a \equiv b \rightarrow a \equiv c
\]

\[
_\circ_\leftarrow \{ a \} = J (\lambda \{ b \} \{ c \} _\leftarrow \rightarrow a \equiv b \rightarrow a \equiv c) (\lambda p \rightarrow p)
\]
A short history of equality

Uniqueness of Identity Proofs?

**Question**
Can all pattern matching proofs done using the eliminator?

**UIP**
Can we prove that all identity proofs are equal?

\[
\text{uip} : (p \ q : a \equiv b) \rightarrow p \equiv q
\]

\[
\text{uip refl refl} = \text{refl}
\]
A short history of equality

Groupoids

A groupoid is a category where every morphism is an isomorphism.

- Categories are the generalisation of preorders and monoids.
- Groupoids are the generalisation of equivalence relations and groups.
A short history of equality

Groupoid laws

Laws

\[ \text{refl} \circ p \equiv p \]
\[ p \circ \text{refl} \equiv p \]
\[ p \circ (q \circ r) \equiv (p \circ q) \circ r \]
\[ p \circ p^{-1} \equiv \text{refl} \]
\[ p^{-1} \circ p \equiv \text{refl} \]

Using only \( J \) we can establish the groupoid laws.

\[ \rho : (p : a \equiv b) \rightarrow p \circ \text{refl} \equiv p \]
\[ \rho = J (\lambda \; p \rightarrow p \circ \text{refl} \equiv p) \, (\lambda \; \{ \_ \} \rightarrow \text{refl}) \]
Groupoids form a model of Type Theory in which $uip$ doesn’t hold. Hence $uip$ is not derivable from $J$ only.
Consider the functions

\[ f : \mathbb{N} \rightarrow \mathbb{N} \]
\[ f = \lambda n \rightarrow n + 0 \]
\[ g : \mathbb{N} \rightarrow \mathbb{N} \]
\[ g = \lambda n \rightarrow n \]

We can show

\[ \text{exteq} : (n : \mathbb{N}) \rightarrow f \ n \equiv g \ n \]
\[ \text{exteq} \ n = \text{add0lem} \ n \]

but we cannot show

\[ \text{eq} : f \equiv g \]

because if such a proof exists.
Then there is one in normal form \((\text{refl})\).
And \(f\) and \(g\) would have to be convertible (same normal form).
However, \(n + 0\) and \(n\) are not convertible.
This shows that the principle:

\[
\text{ext} : (f \ g : A \to B) \\
\to ((a : A) \to f \ a \equiv g \ a) \to f \equiv g
\]

is not provable in Type Theory.
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.
Setoids

- Setoids are sets with an equivalence relation.

\[\text{record Setoid : } Set_1 \text{ where}
\]

\[\text{field}
\]

\[\text{set : Set}
\]

\[\text{eq : set } \rightarrow \text{ set } \rightarrow \text{ Prop}
\]

\[\ldots\]

- 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.

\[
\text{record } _\Rightarrow set_ (A \ B : \text{Setoid}) : \text{Set} \text{ where}
\]
\[
\text{field}
\]
\[
app : \text{set } A \rightarrow \text{set } B
\]
\[
\text{resp} : \forall \{ a \} \ \{ a' \} \rightarrow \text{eq } A \ a \ a' \rightarrow \text{eq } B (\text{app } a) (\text{app } a')
\]

- Equality between functions is extensional equality:

\[
_\Rightarrow _\Rightarrow : \text{Setoid} \rightarrow \text{Setoid} \rightarrow \text{Setoid}
\]
\[
A \Rightarrow B = \text{record } \{
\text{set } = A \Rightarrow \text{set } B;
\text{eq } = \lambda f \ f' \rightarrow \\
\quad \forall \{ a \} \rightarrow \text{eq } B (\text{app } f \ a) (\text{app } f' \ a)\}
\]
Eliminating extensionality

- Adding principles like $\text{ext}$ as constants destroys basic computational properties of Type Theory.
- E.g. there are natural numbers not reducible to a numeral.
- We can eliminate $\text{ext}$ by translating every type as a setoid see my LICS 99 paper: *Extensional Equality in Intensional Type Theory*.
- This construction only works for a proof-irrelevant equality (UIP holds).
Equality of types

- When should two types be provably equal?
- All operations in Type Theory preserve isomorphisms.
- Unlike Set Theory, e.g. \( \{0, 1\} \simeq \{1, 2\} \) but
  \( \{0, 1\} \cup \{0, 1\} \not\simeq \{0, 1\} \cup \{1, 2\} \).
- Indeed, isomorphic types are propositionally indistinguishable in Type Theory.
- Leibniz principle: isomorphic sets should be equal!?
Univalent Type Theory

- Vladimir Voevodsky proposed a new principle for Type Theory: the univalence principle.
- This is inspired by models of Homotopy theoretic models of Type Theory.
- He defines the notion of \textit{weak equivalence} of types.

Voevodsky’s Univalence Principle

Equality of types is weakly equivalent to weak equivalence

- Using this principle we can show that isomorphic types are equal.
- It also implies \textit{ext}.
- However, it is incompatible with \textit{uip}.
The question

- Can we eliminate univalence?
- We cannot use setoids because they rely on UIP.
- Groupoids are better.
- But Groupoids still rely on proof-irrelevance for the equality of equality proofs . . .
- Hence we need $\omega$-groupoids.
- Since the equalities are not all strict we need weak $\omega$-groupoids.
What are weak $\omega$-groupoids?

- There are a number of definitions in the literature, e.g. based on contractible globular operads.
- We need to formalize them in Type Theory . . .
- Formalizing the required categorical concepts creates a considerable overhead.
- Also it is not always clear how to represent them in the absence of UIP.
- E.g. what are strict $\omega$-groupoids?
Globular sets

We define a *globular set* $G : \text{Glob}$ coinductively:

$$
\begin{align*}
\text{obj}_G & : \text{Set} \\
\text{hom}_G & : \text{obj}_G \to \text{obj}_G \to \infty \text{Glob}
\end{align*}
$$

Given globular sets $A, B$ a morphism $f : \text{Glob}(A, B)$ between them is given by

$$
\begin{align*}
\text{obj}_f & : \text{obj}_A \to \text{obj}_B \\
\text{hom}_f & : \prod a, b : \text{obj}_A. \\
& \quad \text{Glob}(\text{hom}_A a b, \text{hom}_B(\text{obj}_f a, \text{obj}_f b))
\end{align*}
$$

As an example we can define the terminal object in $\mathbf{1}_{\text{Glob}} : \text{Glob}$ by the equations

$$
\begin{align*}
\text{obj}_{\mathbf{1}_{\text{Glob}}} & = \mathbf{1}_{\text{Set}} \\
\text{hom}_{\mathbf{1}_{\text{Glob}}} x y & = \mathbf{1}_{\text{Glob}}
\end{align*}
$$
The Identity Globular set

More interestingly, the globular set of identity proofs over a given set $A$, $\text{Id}^\omega A : \text{Glob}$ can be defined as follows:

$$\text{obj}_{\text{Id}^\omega A} = A$$
$$\text{hom}_{\text{Id}^\omega A} a b = \text{Id}^\omega (a = b)$$
Globular sets as a presheaf

Our definition of globular sets is equivalent to the usual one as a presheaf category over the diagram:

\[
0 \xrightarrow{s_0} 1 \xrightarrow{s_1} 2 \ldots n \xrightarrow{s_n} (n + 1) \ldots
\]

with the globular identities:

\[
\begin{align*}
t_{i+1} \circ s_i &= s_{i+1} \circ t_i \\
t_{i+1} \circ t_i &= s_{i+1} \circ t_i
\end{align*}
\]
A syntactic approach

- When is a globular set a weak $\omega$-groupoid?
- We define a syntax for objects in a weak $\omega$-groupoid.
- A globular set is a weak $\omega$-groupoid, if we can interpret the syntax.
- This is reminiscent of environment $\lambda$-models.
The syntactical framework

Contexts

Con : Set

ε : Con

(Γ, C) : Con

Categories

Γ : Con

Cat Γ : Set

C : Cat Γ

a, b : Obj C

C[a, b] : Cat Γ

Objects

• : Cat Γ

C : Cat Γ

Obj C, Var C : Set
Interpretation

1. An assignment of sets to contexts:

\[
\Gamma : \text{Con} \\
\downarrow \quad \downarrow \\
\Gamma : \text{Set}
\]

2. An assignment of globular sets to category expressions:

\[
C : \text{Cat} \\
\gamma : \Gamma \\
\downarrow \\
\text{Glob}
\]

3. Assignments of elements of object sets to object expressions and variables

\[
C : \text{Cat} \\
A : \text{Obj} \\
\gamma : \Gamma \\
\downarrow \\
\text{obj}_{\Gamma} \\
\gamma
\]

subject to some (obvious) conditions such as:

\[
[[\bullet]] \gamma = G \\
[[C[a, b]]] \gamma = \text{hom}_{\Gamma} ([[a]] \gamma) ([[b]] \gamma)
\]
Composition

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow \alpha \\
    & \quad f' \quad g' \\
\end{align*}
\]

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow \beta \\
    & \quad g' \quad f' \\
\end{align*}
\]

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow \gamma \\
    & \quad \alpha' \\
\end{align*}
\]

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow \delta \\
    & \quad \beta' \quad \gamma \\
\end{align*}
\]

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow \beta \cdot \alpha \\
    & \quad \beta' \cdot \alpha' \\
\end{align*}
\]

\[
\begin{align*}
    a & \xrightarrow{f} b \xrightarrow{g} c \\
    & \quad \Downarrow f'' \\
\end{align*}
\]
Telescopes

A telescope $t : \text{Tel } C n$ is a path of length $n$ from a category $C$ of to one of its (indirect) hom-categories:

$$
\frac{C : \text{Cat } \Gamma \quad n : \mathbb{N}}{\text{Tel } C n : \text{Set}}
$$

We can turn telescopes into categories:

$$
\frac{t : \text{Tel } C n}{C 
\vdash t : \text{Cat } \Gamma}
$$
Formalizing composition

\[
\frac{\alpha : \text{Obj}(t \downarrow)}{\beta : \text{Obj}(u \downarrow)} \Rightarrow \frac{\beta \circ \alpha : \text{Obj}(u \circ t \downarrow)}
\]

is a new constructor of \text{Obj} where

\[
t : \text{Tel } (C[a, b]) \quad n \quad u : \text{Tel } (C[b, c]) \quad n \\
\frac{u \circ t : \text{Tel } (C[a, c])}
\]

is a function on telescopes defined by cases

\[
\bullet \circ \bullet C = \bullet \quad u[a', b'] \circ t[a, b] = (u \circ t)[a' \circ a, b' \circ b]
\]
Laws

For example the left unit law in dimension 1:

\[ \text{id}_b \circ f = f \]  

(1)

and in dimension 2.

\[ \text{id}^2_b \circ \alpha = \alpha \]

where \( \text{id}^2_b = \text{id}_{\text{id}_b} \)

In the strict case the 2nd equation only type-checks due to the first. In the weak case we have to apply the previous isomorphism explicitly.
Weak $\omega$-groupoids

Coherence

Example:

\[
\begin{align*}
&(g \ id_b) f \quad (g \ id_g \ f) \\
\rho \ id_f &\quad \alpha_{g, id_g, f} \\
\end{align*}
\]

In summary and full generality:

*For any pair of coherence cells with the same domain and target, there must be a mediating coherence cell.*
Formalizing coherence

\[ x : \text{Obj } C \]
\[ \text{hollow } x : \text{Set} \]

\[ \text{hollow } (\lambda \_\_ ) = \top \ldots \]

\[ f \ g : \text{Obj } C[a, b] \quad p : \text{hollow } f \quad q : \text{hollow } g \]
\[ \text{coh } p \ q : \text{Obj } C[a, b][f, g] \]

\[ \text{hollow } (\text{coh } p \ q) = \top \]
To be able to eliminate univalence we want to interpret Type Theory in a weak $\omega$-groupoid in Type Theory.

As a first step we need to define what is a weak $\omega$-groupoid.

Our approach is to define a syntax for objects in a weak $\omega$ groupoid.

A globular set is a weak $\omega$ groupoid if we can interpret this syntax.

See our draft paper for details: A Syntactical Approach to Weak $\omega$-Groupoids
Further work

- The current definition is quite complex - can we simplify it?
- Can we actually show that the identity globular set is a weak $\omega$-groupoid, internalizing results by Lumsdaine and Garner/van de Berg?
- What is a model of Type Theory in a weak $\omega$-groupoid.
- Can we use this construction to eliminate univalence?