Homotopy and Directed Type Theory: a Sample

Dan Doel

October 24, 2011
Type Theory Overview

- **Judgments**

  \[ \Gamma \vdash \text{ctx} \]
  \[ \Gamma \vdash A \text{ type} \]
  \[ \Gamma \vdash M : A \]

- **Families**

  \[ \Gamma, x : A \vdash B(x) \text{ type} \]

- **Inference**

  \[ \frac{\Gamma_1 \vdash J_1}{\Gamma_2 \vdash J_2} \]
Type Theory Overview

Π and Σ

- **Formation**
  \[
  \frac{\Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Pi_{x:A}B(x) \text{ type}} \quad \frac{\Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Sigma_{x:A}B(x) \text{ type}}
  \]

- **Introduction**
  \[
  \frac{\Gamma, x : A \vdash M : B(x)}{\Gamma \vdash (\lambda x : A. M) : \Pi_{x:A}B(x)} \quad \frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B(M)}{\Gamma \vdash (M, N) : \Sigma_{x:A}B(x)}
  \]

- **Elimination**
  \[
  \frac{\Gamma \vdash F : \Pi_{x:A}B(x) \quad \Gamma \vdash M : A}{\Gamma \vdash FM : B(M)} \quad \frac{\Gamma \vdash P : \Sigma_{x:A}B(x)}{\Gamma \vdash \pi_1 P : A} \quad \frac{\Gamma \vdash P : \Sigma_{x:A}B(x)}{\Gamma \vdash \pi_2 P : B(\pi_1 P)}
  \]
Inductive types

- Define a type as built out of constructors:

```haskell
data N : type where
  zero : N
  suc : N -> N
```

- Induction principle

```haskell
induction : (P : N -> type)
  -> P(zero)
  -> ((n : N) -> P(n) -> P(suc n))
  -> (n : N) -> P(n)
```
Type Theory Overview

Identity types

- **Definition**
  
  \[
  \text{data Id } A \ x : A \to \text{type where} \\
  \text{refl} : \text{Id } A \ x \ x
  \]

- **Elimination**
  
  \[\text{J} : (A : \text{type}) \to (x : A) \to (P : (z : A) \to \text{Id } A \ x \ z \to \text{type}) \to P x \ \text{refl} \rightarrow (y : A) \to (eq : \text{Id } A \ x \ y) \to P y \ eq\]

- **Computation**
  
  \[\text{J } A \ M \ P \ PM \ M \ \text{refl} = PM\]

- **Fancy notation:** \(M \sim_{A} N\)
Type Theory Overview

Identity types

- Simplified (but often useful) version of J:
  \[
  \text{subst} : (A : \text{type}) \to (P : A \to \text{type}) \\
  \to (x \ y : A) \to \text{Id} A \ x \ y \\
  \to P \ x \to P \ y
  \]

- Defining property of equality: respected by all predicates

- Very convenient: we needn’t know anything about P to know that it respects equality

- Recurring theme: how far can we extend this respect?
Type Theory Overview

The universe $U$

- **Formation**

  $\Gamma \vdash U : U$
  $\Gamma \vdash S : U$
  $\Gamma \vdash T(S) : U$

- **Introduction**

  $\Gamma \vdash 0, 1, 2 : U$
  $\Gamma \vdash S : U$
  $\Gamma, x : T(S) \vdash F : U$
  $\Gamma \vdash \prod S F : U$

- **No Elimination**

- **Keep in mind:** $\simeq_U$
Type Theory Overview

Set-theoretic model

- Types denote sets — including U
- Inductive types denote appropriate inductively defined sets
- The identity type denotes equality on said sets
  - We expect identities to be propositions.
  - This suggests a second eliminator for identities ...
Type Theory Overview

Axiom K

K : (A : type) \to (x : A)
\to (P : \text{Id} A x x \to \text{type})
\to P \text{ refl} \to (eq : \text{Id} A x x) \to P \text{ eq}

Also called Uniqueness of Identity Proofs (UIP)

Two motivations

- Identities are propositions
- \text{Id} is an (indexed) inductive type generated by \text{refl}

But K is not definable from J
The Homotopy Model
Standard and non-standard models

- Peano Arithmetic formalizes the natural numbers
- Similar to our \( \mathbb{N} \) type earlier
- Induction principle:
  \[
  P(0) \rightarrow (\forall k. P(k) \rightarrow P(1 + k)) \rightarrow \forall n. P(n)
  \]
- Motivation: every natural number is 0 or a successor thereof
  - But is this what it says?
- No, there are non-standard models
The Homotopy Model

J as an induction principle

- J, interpreted similarly, says, “every identity proof is refl.”
  - K also says this, but evidently in a different way
- This is similarly a mistranslation of J
- Admits (∞-)groupoid/homotopy models
The Homotopy Model

$\infty$-groupoids

A groupoid is a category . . .

\[
\begin{align*}
A &: G \\
\text{id}_A &: A \to A
\end{align*}
\]

\[
\begin{align*}
f &: A \to B \\
g &: B \to C \\
g \circ f &: A \to C
\end{align*}
\]

\[
\begin{align*}
\cdots
\end{align*}
\]

. . . in which all elements are invertible

\[
\begin{align*}
f &: A \to B \\
f^{-1} &: B \to A \\
f^{-1} \circ f &= \text{id}_A \\
f \circ f^{-1} &= \text{id}_B
\end{align*}
\]

. . . An $\infty$-groupoid has infinitely many levels of transformations, and equations are expected to hold only up to higher equivalences.
The Homotopy Model

Types as ∞-groupoids

▶ A suspicious coincidence . . .

\[
\begin{align*}
M : A & \quad F : M \simeq_A N & G : N \simeq_A O \\
\text{refl} : M \simeq_A M & \quad \text{trans } FG : M \simeq_A O \\
F : M \simeq_A N & \quad F : M \simeq_A N \\
\text{sym } F : N \simeq_A M & \quad \cdots : (\text{trans } F (\text{sym } F)) \simeq_{M \simeq_A M} \text{refl}
\end{align*}
\]

▶ The above can all be defined using J

▶ Types together with the identity type naturally form a groupoid

▶ Identity types \(M \simeq_A N\) have their own identity types \(F \simeq_{M \simeq_A N} G\) . . .

▶ . . . and equations in general hold only up to higher identity types

▶ So types are naturally ∞-groupoids
The Homotopy Model

Homotopy $n$-types

Sometimes, an $\infty$-groupoid only has finitely many non-trivial levels

Called an $(\infty, n)$-groupoid, or homotopy $n$-type

Groupoids can be seen as 1-types, sets as 0-types, propositions as $-1$-types

- There are “contractible” types, $-2$-types at the low end

The dimension of a type in this regard is definable in type theory
The Homotopy Model
Homotopy n-types

Contractible : type -> type
Contractible A = Σ( x : A ). Π( y : A ). (Id A x y)

Proposition : type -> type
Proposition A = Π( x y : A ). Contractible (Id A x y)

Type : ℕ -> type -> type
Type zero A = Π( x y : A ). Proposition (Id A x y)
Type (suc n) A = Π( x y : A ). Type n (Id A x y)
The Homotopy Model

Homotopy n-types

- Homotopy $-2$-types are trivial
  - There is an element such that all elements are equivalent to it
- A type is a homotopy $(n + 1)$-type if its identity types are homotopy $n$-types
  - Elements (proofs) of a proposition are trivially equivalent to each other (proof irrelevance)
  - Equality of elements of sets is a proposition
  - Objects of a groupoid have sets of isomorphisms between them.
- ...
The Homotopy Model

Conclusion

- Intuitionistic type theory already admits this higher-dimensional model
  - This model is incompatible with the K axiom, however
- Our earlier “standard” model treated U as a set. . . .
- However, without an inductive eliminator, there is nothing stopping U from being modeled as a higher dimensional type!
  - A groupoid of sets
- U is not provably higher dimensional in standard type theory, of course
The Univalence Axiom

- The traditional model of type theory led us to $K$
- What does the homotopy model suggest?
  - $U$ should be higher dimensional, how can we get there?
The Univalence Axiom

Equivalence

- We want inhabitants of $U$ to be equivalent if there is an isomorphism between them.
- This is typically defined by the following progression:

  $\text{IsEquiv} : (f : S \to T) \to \text{type}$

  $S \simeq T = \Sigma_{f : S \to T} \text{IsEquiv}(f)$

  $\text{substEqv} : S \simeq_U T \to S \simeq T$

  $\text{univalence} : \text{IsEquiv}(\text{substEqv})$

- $\text{substEqv}$ being an equivalence implies that there is an inverse from $S \simeq T$ to $S \simeq_U T$
The Univalence Axiom

Consequences

- Isomorphism of sets implies identity
  - \( Vec \, A \, n \simeq Vec \, A \, m \rightarrow n \simeq_{\mathbb{N}} m? \)
- Univalence has been shown to imply extensionality of functions
  \[
  (\prod_{x:A} f \, x \simeq_B g \, x) \rightarrow f \simeq_{A \rightarrow B} g
  \]
Higher Inductive Types

- We can define new sets via generators using inductive types
- Why not define new n-types?

```haskell
data Circle : type where
  base : Circle
  loop : Id Circle base base

ind-Circle :
  (P : Circle -> type)
  -> (p : P base)
  -> (eq : Id (P base) (subst loop p) p)
  -> (c : Circle) -> P c
```
Benefits

▶ Mathematical
  ▶ Working up to equivalence is common mathematical practice, handled automatically by homotopy type theory
  ▶ Intuitionistic type theory is probably the best direct formulation of $\infty$-groupoids known

▶ “Practical”
  ▶ Functional extensionality is a useful proof principle for reasoning about programs
  ▶ Equivalence-implies-identity aids in code reuse and abstraction
    ▶ List $A$ and $\Sigma_{n: \mathbb{N}}. \text{Vec} A n$ are isomorphic implementations of lists, so any construction on one automatically functions for the other
    ▶ Abstract types and views can be related by equivalence, allowing one to program and prove via the view, while a more efficient abstract type is used internally
Directed Type Theory

- Homotopy type theory generalizes from sets to $\infty$-groupoids
- We can also generalize from groupoids (symmetric) to categories (directed)
  - Instead of $\simeq_A$ with refl, trans, subst, \textit{sym} . . .
  - . . . we have $\equiv_A$ with id, $\circ$, map
Directed Type Theory

Some Details

- Contexts must now track variances:

  \[ \Gamma \vdash \text{ctx} \quad \Gamma \vdash A \text{ type} \quad \Gamma^{\text{op}} \vdash A \text{ type} \]
  \[ \Gamma^{\text{op}} \vdash \text{ctx} \quad \Gamma, x : A^{+} \vdash \text{ctx} \quad \Gamma, x : A^{-} \vdash \text{ctx} \]

  \[ \Gamma, x : A^{-} \vdash B(x) \text{ type} \quad \Gamma, x : A^{-} \vdash M : B(x) \]
  \[ \Gamma \vdash \Pi_{x : A}.B(x) \text{ type} \quad \Gamma \vdash (\lambda x : A. M) : \Pi_{x : A}.B(x) \]

- map acts in response to variance

  \[ \Gamma, x : A^{+} \vdash B(x) \text{ type} \quad \Gamma \vdash \alpha : M \rightarrow_{A} N \]
  \[ \Gamma \vdash \text{map}_{x : A^{+}.B(x)} \alpha : B(M) \rightarrow B(N) \]
Directed Type Theory

Benefits

- Directed types allow an even larger class of transformations to be automatically respected by a large number of constructions
  - Sets and functions
  - Contexts and variable renaming
  - Lambda terms and reduction
- Programming/proving with views and abstract types now needn’t require an equivalence between view and implementation
- Higher dimensional directed type theory has the tools for talking about naturality within the language, and may be able to internally support ‘free’ theorems
Caveats

- There is still work to be done in these areas
  - The univalence axiom has only been postulated thus far; its computational behavior is an open question
    - Licata and Harper have shown canonicity for a 2-dimensional directed theory, but the approach is different
  - Proper hom types have yet to be worked out
    - Instead of \( \text{Id} A \times y \), Hom \( A \times y \)
    - Composition of Hom \( A \times y \) with Hom \( A y z \) has \( y \) in both covariant and contravariant positions
    - Directed type theory works around the issue for now
Further Reading

- The Homotopy Type Theory website
  homotopytypetheory.org

- Univalent Foundations (Voevodsky)
  math.ias.edu/~vladimir/Site3/Univalent_Foundations.html

- Directed Type Theory (Licata and Harper)
  www.cs.cmu.edu/~drl/pubs.html
  www.cs.cmu.edu/~rwh/papers.htm