Dependent type theory as the initial category with families
Internship at Chalmers University of Technology, with Peter Dybjer and Thierry Coquand

Simon Castellan

December 8, 2014
Introduction

Initiality:

- a term of ring theory (eg. $1 + 1$) $\rightarrow$ a unique object in any ring.
- a simply typed $\lambda$-term $\rightarrow$ a unique object in a CCC

Goal: extension of this result to dependent type theory

- Main problem: several derivations for a typing judgement $\rightarrow$ coherence problem

Contribution: an original way of solving this problem
Overview

Coherence problem already solved by [Str91] and [Cur93].

Streicher’s way:

1. Define an annotated syntax
2. Solve the coherence problem there
3. Prove the equivalence with the usual syntax.

Problem with this approach:

1. Definition on untyped terms
2. Annotations are *ad-hoc*.

Our way:

1. Define a *fully* annotated syntax
2. Solve *completely* the problem (as in [Cur93], but less technical)
3. Prove the equivalence.
Table of contents

The calculus (with annotations)

Coherence property

Semantics

The calculus (without annotations)
Martin-Löf’s Logical Framework

- Extension of simply type $\lambda$-calculus with *dependant types*, namely:
  - dependent product: $\prod(x : A)B$ or $\prod(A, B)$
  - universe: type set and a decoding function $el(x)$.
  - polymorphism: $\prod(x : set)(el(x) \Rightarrow el(x))$

- Extends Curry-Howard to first order predicate logic
- Terms appear in types (via $el$) $\Rightarrow$ computation at the level of types
- Type casting: $t : A$ and $A = A'$ then $t : A'$
- Typing judgement $\Gamma \vdash t : A$ along with equality judgement $\Gamma \vdash t = t' : A$
Explicit substitutions

Application for dependent product

\[
\frac{\Gamma \vdash t : \Pi(x : A)B \quad \Gamma \vdash u : A}{\Gamma \vdash t \ u : B\{u/x\}}
\]

⇒ Substitutions becomes part of the syntax.

- **Substitution**: \(\Gamma \vdash f : \Delta \) “\( f \) implements \( \Delta \) in \( \Gamma \)”.
- **Key operations of substitutions**:
  1. **projection**: \( \Gamma \cdot A \vdash p : \Gamma \)
  2. **extension**: \( f : \Gamma \rightarrow \Delta \) and \( \Gamma \vdash t : A \rightarrow \langle f, a \rangle : \Gamma \rightarrow \Delta \cdot A \)
- **Contravariance**: \( \Delta \vdash t : A + \Gamma \vdash f : \Delta \Rightarrow \Gamma \vdash t[f] : A[f] \).
How much annotations

Traditional typing rule:

\[
\frac{\Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(t) : A \rightarrow B}
\]

\(\Gamma, A, B\) are implicit. Fully explicit rule:

\[
\frac{\Gamma \vdash \Gamma \vdash A \quad \Gamma \cdot A \vdash B \quad \Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Gamma, A, B, t) : A \rightarrow B}
\]

▷ Less space for derivations.
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Type constructors:

- set(Γ) (universe)
- Π(Γ, A, B) (dependent product without variable)
- A[f]Γ

Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Type constructors:
- set(Γ) (universe)
- Π(Γ, A, B) (dependent product without variable)
- A[f]_Δ

Typing rule for dependent product

\[
\frac{\Gamma \vdash \Gamma \vdash A \quad \Gamma \cdot A \vdash B}{\Gamma \vdash \Pi(Γ, A, B)}
\]
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Type constructors:
- set(Γ) (universe)
- Π(Γ, A, B) (dependent product without variable)
- A[f]_Γ

Typing rule for substitutions on types

\[
\Gamma \vdash \Delta \vdash \Delta \vdash A \quad \Gamma \vdash f : \Delta
\]

\[
\Gamma \vdash A[f]
\]
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Term constructors:
- $\lambda(\Gamma, A, B, t)$ (λ-abstraction)
- $\text{ap}(\Gamma, A, B, t)$ (unary application)
- $q(\Gamma, A)$ (zeroth de Bruijn variable)
- $(t : A)[f]_{\Delta}$ (substitution)
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Term constructors:

- $\lambda(\Gamma, A, B, t)$ (\(\lambda\)-abstraction)
- $\text{ap}(\Gamma, A, B, t)$ (unary application)
- $q(\Gamma, A)$ (zeroth de Bruijn variable)
- $(t : A)[f]^\Gamma_\Delta$ (substitution)

Typing rule for \(\lambda\)-abstraction

\[
\frac{\Gamma \vdash A \quad \Gamma \cdot A \vdash B \quad \Gamma \cdot A \vdash t : B}{\Gamma \vdash \lambda(\Gamma, A, B, t) : \Pi(\Gamma, A, B)}
\]
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Term constructors:
- $\lambda(\Gamma, A, B, t)$ (\(\lambda\)-abstraction)
- $\text{ap}(\Gamma, A, B, t)$ (unary application)
- $q(\Gamma, A)$ (zeroth de Bruijn variable)
- $(t : A)[f]_{\Delta}$ (substitution)

Type casting

$$
\Gamma = \Gamma' \vdash \Gamma \vdash A = A' \quad \Gamma \vdash t : A \\
\Gamma' \vdash t : A'
$$
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Term constructors:

- \( \lambda(\Gamma, A, B, t) \) (\(\lambda\)-abstraction)
- \( \text{ap}(\Gamma, A, B, t) \) (unary application)
- \( q(\Gamma, A) \) (zeroth de Bruijn variable)
- \( (t : A)[f]_\Delta \) (substitution)

Term equality (\(\beta\))

\[
\frac{\Gamma \cdot A \vdash t : B}{\Gamma \cdot A \vdash t = \text{ap}(\lambda(t)) : B}
\]
Syntax of our calculus

- 8 judgements: typing and equality for contexts, types, terms, substitutions.

Term constructors:

- \( \lambda(\Gamma, A, B, t) \) (\( \lambda \)-abstraction)
- \( \text{ap}(\Gamma, A, B, t) \) (unary application)
- \( q(\Gamma, A) \) (zeroth de Bruijn variable)
- \( (t : A)[f]^{\Gamma}_{\Delta} \) (substitution)

Term equality (\( \eta \))

\[
\Gamma \vdash t : \Pi(\Gamma, A, B) \\
\Gamma \vdash t = \lambda(\text{ap}(t)) : \Pi(\Gamma, A, B)
\]
Compressing derivations

- $\delta \mapsto \delta^z$: compressing derivations by
  1. transitivity of equality

\[
\begin{array}{c}
\vdash \Gamma'' \vdash A \\
\Gamma' = \Gamma'' \vdash \\
\vdash \Gamma' \vdash A \\
\Gamma = \Gamma' \vdash \\
\vdash \Gamma \vdash A \\
\end{array}
\]

- $\Gamma' \vdash A$ \quad $\Gamma = \Gamma' \vdash$ \quad $\Gamma'' \vdash A$ \quad $\Gamma = \Gamma''$ \quad $\Gamma \vdash A$

2. reflexivity

\[
\vdash \Gamma \vdash A \\
\Gamma = \Gamma \vdash \\
\vdash \Gamma \vdash A
\]

Theorem
Let $\delta$ and $\delta'$ be two derivations of a judgement $J$. We have $\delta^z \equiv \delta'^z$. 
Coherence lemma

Goal: a definition on derivations → definition on judgements.

Interpretation: A map \( \varphi : D \rightarrow X \) such that

\[
\varphi \left( \frac{\delta : \Gamma \vdash t : A \quad \Gamma \vdash A = A'}{\Gamma \vdash t : A'} \right) = \varphi(\delta)
\]

Theorem

Any interpretation \( \varphi : D \rightarrow X \) defined on derivations yields a map \( \varphi : J \rightarrow X \) defined on typing judgements such that whenever \( \delta : J \) then \( \varphi(\delta) = \bar{\varphi}(J) \)
Categories with families (CwF)

- Categorical semantics centered around contexts and substitutions as morphisms between contexts: definitional equality becomes equality in a CwF
- Category of CwFs
- Example: term model $\mathbb{T}$: quotient of syntax by definitional equality.
- Goal: initiality of $\mathbb{T}$
Initiability of $T$

Let $\mathcal{C}$ be a CwF.

1 Interpretation in any CwF: a map $\llbracket \cdot \rrbracket$ from the syntax to $\mathcal{C}$

\[
\begin{align*}
\delta_{\Gamma} : \Gamma \vdash & \quad \delta_A : \Gamma \vdash A \\
\delta_B : \Gamma \cdot A \vdash B \\
\Gamma \vdash \Pi(\Gamma, A, B) \Rightarrow \Pi(\llbracket \delta_{\Gamma} \rrbracket, \llbracket \delta_A \rrbracket, \llbracket \delta_B \rrbracket)
\end{align*}
\]

2 Extends to a morphism of CwFs: $\llbracket \cdot \rrbracket : T \to \mathcal{C}$

for instance $F(\llbracket \Gamma \vdash \rrbracket) = \llbracket \Gamma \vdash \rrbracket$

3 Uniqueness: there is a unique map from $T$ to $\mathcal{C}$.

$\Rightarrow T$ is an initial object.
Syntax and term model

We now consider the same calculus but without the extra annotations.

Type constructors:

- set (universe)
- $\Pi(A, B)$ (dependent product without variable)
- $A[f]$ (substitution)
Syntax and term model

- We now consider the same calculus but without the extra annotations.

Type constructors:
- set (universe)
- $\Pi(A, B)$ (dependent product without variable)
- $A[f]$ (substitution)
Syntax and term model

- We now consider the same calculus but without the extra annotations.

Term constructors:

- $\lambda(t)$ (\lambda-abstraction)
- $\text{ap}(t)$ (unary application)
- $q$ (variable)
- $t[f]$ (substitution)
Syntax and term model

- We now consider the same calculus but without the extra annotations.

Term constructors:

- $\lambda(t)$ ($\lambda$-abstraction)
- $\text{ap}(t)$ (unary application)
- $q$ (variable)
- $t[f]$ (substitution)

- $\mathbb{T}^i$: the implicit term model
- Stripping operator $s$ from $\mathbb{T}$ to $\mathbb{T}^i$
- **Goal**: $s : \mathbb{T} \simeq \mathbb{T}^i$
s is one-to-one

- Injectivity of $s$: if $s(\Gamma) = s(\Gamma')$ then $\Gamma = \Gamma' \vdash$.
- hard part, reflexivity case: if $s(\Gamma) \equiv s(\Gamma')$ then $\Gamma = \Gamma' \vdash$.
- We need normalisation, because of the substitution rule:

$$\Gamma \vdash f : \Delta \quad \Delta \vdash t : A \quad \frac{}{\Gamma \vdash t[f] : A[f]}$$

No $\Delta$ in conclusion.

1. Prove the result for normal term which only substitutions in specific situations.
2. Prove that the result extend to non-normal terms.

- $s$ has an inverse $T^i \to T$.

1. By induction: build a right inverse $t : T^i \to T$ ($s \circ t = \text{Id}_{T^i}$)
2. By initiality of $T$, we know that $t \circ s = \text{Id}_T$

$\to T^i$ is initial.
Conclusion

- Original method: fully annotated syntax
- Extension to other dialects (and GAT)
- Third initial CwF: semantic domain (normalization by evaluation)
P.L. Curien.
Substitution up to isomorphism.

T. Streicher.
*Semantics of type theory: correctness, completeness, and independence results.*