Formalising semantics of dependent type theory in dependent type theory

(work in progress)

Peter LeFanu Lumsdaine
joint work with Håkon Gylterud, Erik Palmgren

DMV Hamburg, 25 September 2015

Modelling DTT

Early models of dependent type theory: constructed by hand.

Construction: multiple large mutual inductions over syntax—types, terms, judgement derivations. . .

Many moving parts to deal with: interaction with substitution, α-conversion, . . .

But: structure of construction similar for many models. Redundancy, duplication of effort!

Algebraic semantics

Algebraic semantics (Cartmell and followers): aim to abstract away the common structure of models.

Have algebraic structure, *category with attributes* (or variants), encoding common structural core of DTT. Then (template): define extra operations on a CwA corresponding to desired logical connectives, and prove:

Theorem (Cartmell, Streicher, Hofmann, . . .)

The syntax of dependent type theory with logical connectives XYZ forms a CwA with XYZ-structure; and in fact this is the initial CwA with XYZ-structure.

Encapsulates the big induction proof once and for all.

Now any CwA with XYZ-structure carries canonical interpretation of syntax. So XYZ-CwA's give good notion of *model of DTT with XYZ.*

Algebraic semantics

Very convenient technique. Since then: most (denotational) models of DTT constructed along these lines. (Streicher, Hofmann, Hofmann-Streicher, Coquand et al, Voevodsky, etc.)

Also, various good theorems provable using CwA's and relatives: conservativity of logical framework presentation (Hofmann), coherence theorems (Hofmann, Voevodsky, Lumsdaine–Warren), etc.

Lots of good work done with this setup.

Everything in the garden is lovely?

Actually: situation not quite so satisfactory.

Dissatisfactions

Most obviously: no *general* theorem.

In practice: "everyone knows" straightforward to extend definitions and theorem to any reasonable logical rules.

General Belief

Any reasonable logical rules correspond to certain struture on CwA's; and the snytactic CwA of DTT with those rules is the initial CwA with that structure.

But precise general statement: difficult to formulate! What even are "reasonable" rules?

> "
>
> I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description, and perhaps I could never succeed in intelligibly doing so. But I know it when I see it [...]"
>
> — Potter Stewart, U.S. S.C.J., *Jacobellis v. Ohio* 1964

Dissatisfactions

More surprisingly: even *specific* cases hard to find/give.

Only 2 detailed proofs in literature (as far as I can find): Streicher Habilitationthesis, Hofmann Thesis.

Various other sketches; most contain (minor) errors.

> “
>
> People say that de Bruijn indices and explicit substitutions are difficult to implement. I agree, I spent far too long debugging my code. But because every bug crashed and burned my program immediately, I at least knew I was not done. In contrast, “manual” substitutions hide their bugs really well, and so are even more difficult to get right.”
>
> — Andrej Bauer, *How to implement DTT, III*

Extending: conceptually straightforward, but quite intricate. Should we be comfortable saying “straightforward”?

Long-term goals

Goal: generalise setting and theorems. Define reasonable class of type theories, and corresponding algebraic structures.

Not hard to make proposals; hard to be sure they're right.

Fit-for-purpose test: can we generalise theorems, esp. the initiality theorem?

"Warm-up": really get to know the existing proofs of specfic cases. How?

> "
>
> Formalise, formalise, formalise! (Only be sure always to call it please *research*.)"
>
> — Tom Lehrer, *Lobachevsky* (adapted)

Short-term goals

Goal: formalise the initiality theorem, for a specific small-ish type theory. Roughly, formalise Streicher's result and proof.

(Also: examples of CwA's. But today I'll focus on initiality theorem.)

Formalise in what? In Coq—in dependent type theory!

Explanation 1: MLTT/CIC our preferred general foundation.

Explanation 2: "When you have a hammer, everything looks like a nail."

Secondary payoff of formalisation: forkability. Even with just small core formalised, other authors can adapt to larger type theories as needed. Referees can verify "straightforward extension" without re-checking whole proof by hand.

Gödel?

FAQ: doesn't Gödel say this is impossible (unless TT inconsistent)?

Answer 0: No!

Answer 1: Consider situation with ZFC. Can formalise the meta-theory (proof theory, model theory) of arbitrary first-order theories, including ZFC itself. Just can't prove models of ZFC exist (unless it's inconsistent).

Answer 2: even if you want model existence, don't need to fundamentally change meta-theory. Just need extra assumptions (e.g. universe existence).

Overall project map

Object theory: for now, just DTT with function types, one base type. Aim: extend later.

Meta-theory: CIC, but not using *Prop*: i.e. DTT with function types, inductive types, (predicative) universes. (Probably one universe enough.) Aim: keep fixed!

Five main components:

1. syntax [done];
2. corresponding algebraic structures [done];
3. interpretation function [in progress];
4. syntactic category;
5. initiality.

Design decisions 1

How to formalise syntax?

Nothing fancy! As bricks-and-mortar as possible.

- Raw expressions, with typing judgements afterwards. NOT inductive-inductive, nominal, HOAS etc.
- Raw expressions as labelled trees, not "parseable strings of symbols".
- Named variables/identifiers, not de Bruijn indices. (Precisely: type V of variables/identifiers, assumed infinite and with decidable equality.)
- Full annotation: e.g. $\mathrm{app}_{A,B}(f, a)$, not just $\mathrm{app}(f, a)$.

Guiding principle: does it fit how we think of syntax when using it?

Design decisions 2

How to formalise algebraic semantics?

Definition

(Classical.) A *category with attributes*:

- category $\mathbf{C}$;
- functor $\mathrm{Ty} : \mathbf{C}^{\mathrm{op}} \to \mathrm{Set}$;
- for $\Gamma \in \mathrm{ob}\,\mathbf{C}$, $A \in \mathrm{Ty}(\Gamma)$, object and map $\pi_A : \Gamma.A \to \Gamma$;
- for $f : \Gamma' \to \Gamma$, $A \in \mathrm{Ty}(\Gamma)$, map $q(f, A) : \Gamma'.A[f] \to \Gamma.A$, exhibiting $\pi_{A[f]}$ as pullback of π_A along f;
- a distinguished object $\mathbf{1}$ (optional).

Design decisions 2

We use *E-categories with attributes*: roughly, CwA's based on *setoids*.

Definition

As a CwA, but

- ob $\mathbf{C}$ an arbitrary type;
- all other sets become setoids (e.g. hom-setoids $\mathbf{C}(\Gamma', \Gamma)$);
- maps between setoids respect setoid equalities;
- context extension becomes functor $D(\mathrm{Ty}(\Gamma)) \to \mathbf{C}/\Gamma$.

Cons, compared to HoTT (pre-)categories: A bit more work in some spots, e.g. explicitly stating how dependent operations respect equality.

Pros: Very foundation-agnostic: interpretable in both HoTT (with univalence, non-set categories) and classical foundations (with UIP). Very constructive: no quotients etc.

Streicher's proof

Streicher: constructs interpretation in two stages.

First: define a *partial* interpretation on "raw judgements". (By induction on raw syntax.)

E.g. for a "raw type judgement" $\Gamma \vdash A$, give a (possibly-defined) *semantic context* and *semantic type* over it, i.e.

$$
\begin{aligned}
[\![B_1]\!] &\in \mathrm{Ty}(\mathbf{1}), \\
[\![B_2]\!] &\in \mathrm{Ty}(\mathbf{1}.[\![B_1]\!]), \\
&\vdots \\
[\![B_n]\!] &\in \mathrm{Ty}(\mathbf{1}.[\![B_1]\!].\cdots.[\![B_{n-1}]\!]), \\
[\![A]\!] &\in \mathrm{Ty}(\mathbf{1}.[\![B_1]\!].\cdots.[\![B_n]\!]).
\end{aligned}
$$

Second: prove that for derivable judgments, this is defined. (By induction on derivations.)

Novelty 1

We split into three stages, not just two:

1. A priori, give *multi-valued* function (neither uniqueness nor existence of values assumed);
2. then prove uniqueness, giving a *partial* function;
3. then prove existence, giving a function.

Implementation: define operations M, P so that:

- multi-valued functions $A \multimap B$ are functions $A \to M(B)$;
- partial functions $A \rightharpoonup B$ are setoid maps $A \to P(B)$.

In fact: $M(-)$ and $P(-)$ form monads, in the functional programming sense! Very congenial to program with.

Novelty 2

Instead of interpreting whole raw judgements, we interpret the principal part of a judgement, given intended interpretations of the presuppositions.

"$\Gamma \vdash A$ type." For a raw type expression A, and any semantic context[1] Γ, have a (multi-/partial-) type $[\![A]\!]_\Gamma \in \mathrm{Ty}(\Gamma)$

"$\Gamma \vdash t : A$." For a raw term expression t, a semantic context Γ, and (semantic) type $A \in \mathrm{Ty}(\Gamma)$, have a (multi-/partial-) section $[\![t]\!]_{\Gamma,A}$ of $\pi_A : \Gamma.A \to \Gamma$.

- Allows interpretation to be structurally inductive on single raw expressions.
- Reduces use of equality of semantic contexts, and resultant coherence isomorphisms.

[1] actually not exactly; see next slide

Novelty 3

Actually: we throw out semantic contexts entirely!

Interpret syntactic contexts as objects equipped with *environments*.

Definition

An *environment* on $\Gamma \in \mathrm{ob}\,\mathbf{C}$: a finite partial map from V to pairs (A, t), where $A \in \mathrm{Ty}(\Gamma)$, and t is a section of π_A.

So: for raw type A, and $\Gamma \in \mathrm{ob}\,\mathbf{C}$, $E \in \mathrm{Env}(\Gamma)$, interpretation is a (multi-/partial-) $[\![A]\!]_{\Gamma,E} \in \mathrm{Ty}(X)$.

- Further reduces equalities, coherence isomorphisms.
- Simplifies reindexing/substitution lemmas: environments can be reindexed.
- A “just right” abstraction: carries exactly the information used in interpreting an expression. (Environment is consulted just when expression is a variable.)

Local road map

Outline of interpretation construction (mostly but not entirely done):

1. Multi-valued interpretation. (Induction on expressions.)
2. Stability under reindexing, environment extension, and equality of semantic arguments. (Induction on expressions; inextricably mutual.)
3. Behaviour under substitution into expressions. (Induction on expressions.)
4. Uniqueness: partial interpretation. (Induction on expressions.)
5. Partial interpretation of (syntactic) contexts, as objects-with-environments. (Induction on contexts.)
6. Definedness: full interpretation of well-typed judgements. (Induction on derivation.)

Summary

Payoffs

- Well-developed libraries on CwAs and syntax.
- Forkable/extendable proof of interpretation theorem (and eventually initiality).
- Examples of CwA's.
- Better understanding towards generalisation.

Current conclusions

- Yeeeees…theorem should extend "straightforwardly" (if laboriously) to "reasonable type theories".
- But: direct approach probably not feasible for general statement. Need to go via intermediate abstractions.