Formalizations of Metatheoretical Properties of Type Theories

Jonas Oberhauser

Programming Systems Lab
Saarland University

Joint work with Chad E. Brown

August 17, 2012

Introduction

Goal:
Formalize proofs of confluence and termination of ECC in Coq
This is probably too hard, so we will only go as far as CC-1

- Confluence of Simply Typed Lambda Calculus (Done)
- Termination of Simply Typed Lambda Calculus (Done)
- Formalization of CC (Done by Barras) and CC-1 (Done)
- Extend proofs of confluence and termination to CC-1

CC-1

CC $+ \mathrm{Type}_0$, with type inclusion:

$$\frac{T{:}\mathrm{Prop}}{T{:}\mathrm{Type}_0}$$

However, no Σ

Termination

$$t \rightarrow\!\!\| \iff \forall t', t \rightarrow_\beta t' \;\rightarrow\; t' \rightarrow\!\!\|$$

The set of strongly normalizable (=terminating) terms

$$\mathrm{SN} := \{t \mid t \rightarrow\!\!\|\}$$

Proof by Induction (Fails)

First idea: Prove if t has Type T, $t \not\to\!\!\|$ by induction on $t : T$:

- $n \not\to\!\!\|$ (there is no t' such that $n \to_\beta t'$)
- $t \not\to\!\!\| \;\to\; \lambda T.t \not\to\!\!\|$ (the lambda reduces only when t reduces)
- But: Just because $t, s \not\to\!\!\|$, it doesn't have to be the case that $t\,s \not\to\!\!\|$ (at least not obviously)

Interpretations

So we can't directly prove via induction:
Everything that types terminates
Since Girard we do this instead:

- We define an interpretation of a type T, $[\![T]\!] \subset \mathrm{TERM}$
- We show that this interpretation contains only terminating terms: $[\![T]\!] \subset \mathrm{SN}$
  In fact, we show a little bit more - that the interpretation is a reducibility candidate. Apart from termination, there are two more reducibility candidate conditions that are not important for the sake of this talk.
- We show that if t has type T, then t is in $[\![T]\!]$

These interpretations are sometimes also called logical relations

Interpretations for STLC

Recursive definition on T:

$$[\![\mathrm{B}]\!] := \mathrm{SN}$$

$$[\![S \to U]\!] := \{t \mid \forall s \in [\![S]\!], (t\, s) \in [\![U]\!]\}$$

This will give us the additional assumption that if $t, s \to\!\!\|$, $t\, s \to\!\!\|$

We have formalized the proof for STLC

Problems of richer theories

- Dependencies: Polymorphism, Dependent Types, Type Operators
- Type Hierarchy

Types are Terms

We have to interpret all terms, not just those representing a type. We do so by using meaningless junk-interpretations of which we know that they are reducibility candidates.

Dependencies

In the lambda cube, there are three types of dependencies:

- Types depending on kinds, we call this polymorphism
- Kinds depending on types, we call this dependent types
- Kinds depending on kinds, we call this type operators

Barras told us how to deal with these in 1997. (not personally though)

Dependencies

Polymorphism and Dependent Types : Easy

How to deal with Type Operators?
We annotate terms with type skeletons, which resemble simple types:

$$\mathrm{SKELETON} = \bigstar \mid s_1 \rightarrow s_2$$

For kinds, we approximate the kind by matching type operators in the term with a $\rightarrow$ in the skeleton.

For predicates we use the skeleton of the type of the predicate (which is a kind)

For basic terms we don't need this annotation

$\rightarrow$ Three distinct levels: Kinds, predicates, basic terms

Some Examples

Prop is a kind and has no Π: $\bigstar$

$\Pi p : \mathrm{Prop}.\mathrm{Prop}$ is a kind: $\bigstar \rightarrow \bigstar$

$\lambda p : \mathrm{Prop}.p$ is a predicate and has type $\Pi p : \mathrm{Prop}.\mathrm{Prop}$: $\bigstar \rightarrow \bigstar$

$\Pi t : \mathrm{True}.\mathit{Prop}$ looks like $\bigstar \rightarrow \bigstar$. It's not a type operator, so we remove the gray parts: $\bigstar$

$\lambda t : \mathrm{True}.b$ is a basic term and has no skeleton.

How we use skeletons

We extend the notion of reducibility candidate to candidate of some order S:

If S is $\bigstar$, then $[\![t]\!]$ is a candidate if it is a reducibility candidate. (it's not a type operator so we're pretty close to STLC)

If S is $S_1 \to S_2$, then $[\![t]\!]$ is a candidate if it is a function that maps candidates of order S_1 to candidates of order S_2

$\to$ We are lifting the old argument from types to kinds:

types: map elements of a candidate to elements of a candidate

kinds: map candidates to candidates

Type Hierarchy

Kinds, predicates, basic terms might not be enough:

$$(\lambda x : \mathrm{True}, x) \ : \ (\Pi x : \mathrm{True}, \mathrm{True}) \ : \ \mathrm{Prop} \ : \ \mathrm{Type}_0 \ : \ \mathrm{Kind}$$

$\rightarrow$ four levels

$$(\lambda x : \mathrm{True}, x) \ : \ (\Pi x : \mathrm{True}, \mathrm{True}) \ : \ \mathrm{Type}_0 \ : \ \mathrm{Kind}$$

$\rightarrow$ distinction might become harder

But all of these are just suspicions

What We Will Do

- Take the formalization of termination of CC by Barras
- Go to CC-1
- Extend the proof of termination to account for the additional Type level
- Pray that no problems arise and that if they do we can solve them

Back-Up Plan

There is another proof of termination by Luo which uses saturated sets and a method called quasi-normalization

This proof seems to be much harder to formalize, but works for ECC

If we end up having problems we don't seem to be able to fix, we will formalize quasi-normalization as a first step to termination of ECC

Confluence

We have already formalized the confluence of STLC using Takahashi's method and we will try to extend this proof to CC-1.

This should run through easily

References I

Girard.
Proofs and Types.
1989.

Barendregt.
The Lambda Calculus.
1984.

Luo.
Computation and Reasoning.
1994.

Brown, Smolka.
Semantics Lecture Notes.
2012.

Barras.
Coq in Coq.
1997.