Formalizing Termination of CC

Jonas Oberhauser

Programming Systems Lab
Saarland University

Joint work with Chad E. Brown

May 2, 2013

Introduction

Our Goal:
Formalize Geuver's proof of Termination[2] for CC_{Σ}

What we did:

1. Take Barras' formalization [1] for CC
2. Extend the syntax, definitions and lemmas to CC_{Σ}
3. Take Geuver's definitions and implement them

This was surprisingly easy (took us around three weeks total)

On the Way There

1. Understand Barras' formalization
2. Find a mathematical representation of the proof that is easy to explain
3. Explain Barras' formalization in terms of the mathematical representation

   This was surprisingly hard (took us about a year)

What We Left Out

- Superkinds
- Formalization that we will discuss today
- The concept of Geuvers' proof does not extend easily to ECC

We did not finish these parts before the deadline ran out

Plan for Today

- Recapitulate CC_Σ and Termination
- Show a proof sketch of Geuvers' proof
- Pick two details from Geuvers' proof and discuss a formalization

Warning:

- The formalization we discuss is not the same as in the thesis, but is very close to the mathematical proof
- Many things are oversimplified

The Syntax of CC_Σ

Set of pseudoterms Λ (very similar to Luo[3]):

universe	$\mathrm{Prop}, \mathrm{Type}_0$
variable	$x, y, z, ...$
lambda	$\lambda x : T.t$
application	$(t\, u)$
product	$\forall x : T.U$
pair	$(t, u)_{\Sigma x:T.U}$
projection	$\pi_i(t)$ for $i \in \{1, 2\}$
sigma	$\Sigma x : T.U$

The Typing Rules of CC_Σ

(small excerpt)

$$\textit{product}\quad \frac{\Gamma \vdash T : s_1 \qquad \Gamma, x : T \vdash U : s_2}{\Gamma \vdash (\forall x : T.U) : s_2}$$

$$\textit{sigma}\quad \frac{\Gamma \vdash T : \mathrm{Prop} \qquad \Gamma, x : T \vdash U : \mathrm{Prop}}{\Gamma \vdash (\Sigma x : T.U) : \mathrm{Prop}}$$

$$\textit{pair}\quad \frac{\begin{array}{c}\Gamma \vdash t : T \\ \Gamma \vdash u : U_t^x \qquad \Gamma \vdash T : \mathrm{Prop} \qquad \Gamma, x : T \vdash U : \mathrm{Prop}\end{array}}{\Gamma \vdash (t, u)_{\Sigma x:T.U} : (\Sigma x : T.U)}$$

Reduction

Main reduction rules:

$$\textit{beta}\quad \frac{}{(\lambda x : T.t)\, u \to_\beta t^x_u} \qquad \textit{proj1}\quad \frac{}{\pi_1((t,u)_{\Sigma x:T.U}) \to_\beta t}$$

$$\textit{proj2}\quad \frac{}{\pi_2((t,u)_{\Sigma x:T.U}) \to_\beta u}$$

Congruence Rules

Plus congruence rules that allow reduction in subterms, e.g.:

$$\frac{T \to_\beta T'}{\forall x : T.U \to_\beta \forall x : T'.U}$$

Termination

$$\mathrm{SN} := \{t \| \forall t', t \to_\beta t' \ \to \ t' \in \mathrm{SN}\}$$

Termination: $\Gamma \vdash t : T$ implies $t \in \mathrm{SN}$

General Proof Outline

General proof technique since Tait in 1967[4]:

1. Define interpretation of terms $[\![T]\!]_\xi$
2. Prove that $\Gamma \vdash t : T$ implies $t \in [\![T]\!]_\xi$
3. Prove that $\Gamma \vdash t : T$ implies $[\![T]\!]_\xi \subseteq \mathrm{SN}$
4. Consequently $\Gamma \vdash t : T$ implies $t \in \mathrm{SN}$

Geuvers' Proof

1. Classify terms into disjoint classes: superkind, kind, constructor, object
2. Define a set interpretation $\mathcal{I}_\eta(T)$ to get the structure of superkinds and kinds
3. Define interpretation $[\![T]\!]_\xi$ for superkinds, kinds and constructors

Classes

Superkinds: Type_0 (the only one)

Kinds: Prop, $\forall x : \mathrm{Prop}.\mathrm{Prop}$, $\forall x : \mathrm{True}.\mathrm{Prop}$, ...

Constructors: $\mathrm{True} := \forall x : \mathrm{Prop}.\forall y : x.x$, $\mathrm{False} := \forall x : \mathrm{Prop}.x$, $\neg := \lambda x : \mathrm{Prop}.\forall y : x.\mathrm{False}$, ...

Objects: $I := \lambda x : \mathrm{Prop}.\lambda y : x.y$, ...

Classification

Function $\mathrm{class}_\eta(T) \in \{\mathcal{S}, \mathcal{K}, \mathcal{C}, \mathcal{O}\}$: maps term to its class

Soundness:

- $\mathrm{class}_\eta(\mathrm{Type}_0) = \mathcal{S}$
- $\Gamma \vdash T : \mathrm{Type}_0$ implies $\mathrm{class}_\eta(T) = \mathcal{K}$
- $\Gamma \vdash T : K : \mathrm{Type}_0$ implies $\mathrm{class}_\eta(T) = \mathcal{C}$
- $\Gamma \vdash T : K : \mathrm{Prop}$ implies $\mathrm{class}_\eta(T) = \mathcal{O}$

Set Interpretation

$$\mathcal{I}_\eta(s) := \mathrm{CR}$$

$$\mathcal{I}_\eta(\forall x : T.U) := \mathcal{I}_\eta(T) \to \mathcal{I}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{K}$$

$$\mathcal{I}_\eta(\forall x : T.U) := \mathcal{I}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{C}$$

Where CR is a subset of 2^{SN}

Soundness: $\mathcal{I}_\eta(T)$ is defined for superkinds and kinds (one page proof)

Interpretation

$$
\begin{aligned}
[\![s]\!]_\xi &:= \mathrm{SN} \\
&\dots \\
[\![\lambda x : T.u]\!]_\xi &:= \left(\Phi \in \mathcal{I}_\xi\,(T) \mapsto [\![U]\!]_{\xi^x_\Phi}\right) && \text{if } T \in \mathcal{K}_\xi \\
&\dots \\
[\![t\ u]\!]_\xi &:= [\![t]\!]_\xi([\![u]\!]_\xi) && \text{if } u \in \mathcal{C}_\xi \\
&\dots
\end{aligned}
$$

Soundness: $\Gamma \vdash T : K$ implies $[\![T]\!]_\xi$ is defined and in $\mathcal{I}_\eta\,(K)$ (two and a half pages of proof when omitting many cases)

Problems

Set interpretation is partial

Interpretation is partial, definedness and correctness need to be proven concurrently

Partial functions are hard to formalize in Coq

Solution of Barras

Use inductive skeletons rather than set interpretation

Couple classification and skeletons: class with the option of a skeleton

Define a total procedure `cl_term` that computes class and possibly skeleton

Use skeletons and Coq's dependent types to make interpretation total

Skeletons

Use inductive type:

$$\mathrm{Skel} := \bigstar \mid s_1 \to_S s_2$$

Go to set interpretation later by mapping skeletons to sets:

$$\mathrm{can}(\bigstar) := \mathrm{CR}$$
$$\mathrm{can}(s_1 \to_S s_2) := \mathrm{can}(s_1) \to \mathrm{can}(s_2)$$

Skeletons in Action

Before:

$$\mathcal{I}_\eta(s) := \mathrm{CR}$$
$$\mathcal{I}_\eta(\forall x : T.U) := \mathcal{I}_\eta(T) \to \mathcal{I}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{K}$$
$$\mathcal{I}_\eta(\forall x : T.U) := \mathcal{I}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{C}$$

After:

$$\mathrm{sk}_\eta(s) := \bigstar$$
$$\mathrm{sk}_\eta(\forall x : T.U) := \mathrm{sk}_\eta(T) \to_S \mathrm{sk}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{K}$$
$$\mathrm{sk}_\eta(\forall x : T.U) := \mathrm{sk}_{\eta^x_{\mathrm{class}_\eta(T)}}(U) \quad \text{if } \mathrm{class}_\eta(T) = \mathcal{C}$$

With $\mathrm{can}(\bigstar) = \mathrm{CR}$ and $\mathrm{can}(s_1 \to_S s_2) = \mathrm{can}(s_1) \to \mathrm{can}(s_2)$.

Class With an Option of Skeleton

```
Inductive skel : Type :=
    | PROP : skel
    | PROD : skel -> skel -> skel.

Inductive class : Type :=
    | Obj : class
    | Con : class
    | Knd : skel -> class
    | Snd : skel -> class.
```

Note that this is basically `class + option skel`

Class

Function $\mathrm{class}_\eta\,(t) + \mathrm{sk}_\eta\,(t)$:

```
Fixpoint cl_term (t : term) : cls -> class :=
    fun eta : cls =>
    match t with
    ...
    | Prod T U =>
        let eta_ex := TCs _ (cl_term T eta) eta in
        match cl_term T eta, cl_term U eta_ex with
        | Knd s1, Knd s2 => Knd (PROD s1 s2)
        | _, Knd s => Knd s
        | _, Con => Con
        | _, c1 => c1
        end
    ...
    end.
```

Uses class knowledge to decide whether the term has a skeleton or not; very clever in dealing with partial skeletons.

Decoupling Skeletons and Classification

```
Inductive skel : Type :=
    | PROP : skel
    | PROD : skel -> skel -> skel.

Inductive class : Type :=
    | Obj : class
    | Con : class
    | Knd : class
    | Snd : class.
```

Class is class and skeletons are skeletons

Decoupled Computing

```
Fixpoint cl_term (t : term) : cls -> class :=
    fun eta : cls =>
    match t with
    ...
    | Prod T U =>
        cl_term U ((cl_term T eta) :: eta)
    ...
    end.

Fixpoint skeleton T eta : skel :=
    match T with
    | Prod T U =>
      match cl_term T eta with
      | Knd => PROD (skeleton T eta) (skeleton U (cl_ex T eta))
      | _ => (skeleton U (cl_ex T eta))
      end
    | _ => PROP
    end.
```

Simply gives everything a skeleton.

Formalizing Interpretation

$$
\begin{aligned}
[\![s]\!]_\xi &:= \mathrm{SN} \\
&\dots \\
[\![\lambda x : T.u]\!]_\xi &:= \left(\Phi \in \mathcal{I}_\xi\,(T) \mapsto [\![U]\!]_{\xi^x_\Phi}\right) && \text{if } T \in \mathcal{K}_\xi \\
&\dots \\
[\![t\ u]\!]_\xi &:= [\![t]\!]_\xi([\![u]\!]_\xi) && \text{if } u \in \mathcal{C}_\xi \\
&\dots
\end{aligned}
$$

Problem:
Direct definition of `interpret T xi`, $[\![t]\!]_\xi([\![u]\!]_\xi)$ would require
`interpret t xi : A -> B` and `interpret u xi : A`

How to Interpret Application

Barras' solution:

Return type of `interpret T xi` is dependent

So: `interpret T xi : forall s, Can s`

Formalizing $[\![t]\!]_\xi([\![u]\!]_\xi)$:

`(interpret t xi (PROD s1 s2)) (interpret u xi s1)`

Which Skeleton?

```
interpret (t u) xi s2
  = (interpret t xi (PROD s1 s2)) (interpret u xi s1)
```

Problem: How to get `s1`?

Recall soundness: When $\Gamma \vdash T : K$, $[\![T]\!]_\xi \in \mathcal{I}_\eta(K)$

Therefore, `s1` should be skeleton of a `T` where $\Gamma \vdash u : T$

$\rightarrow$ Get the skeleton of `T` by looking at `u`

Constructor-Skeletons

When `u` is a constructor and `T` is the type of `u`, can we get the skeleton of `T` by looking at `u`?

Barras: Yes! With extended `cl_term`:

- For kinds and superkinds, return class + skeleton
- For constructors, return class + constructor-skeleton
- For objects, return class

$\rightarrow$ decouple this again

Decoupling Constructor Skeletons

```
Fixpoint c_sk t eta sk : skel :=
   match t with
   | Ref n => nth_def _ PROP sk n
   | Abs T u =>
      match cl_term T eta with
      | Knd => PROD (skeleton T eta)
                    (c_sk u (cl_ex T eta) (sk_ex T eta sk))
      | _ => (c_sk u (cl_ex T eta) (sk_ex T eta sk))
      end
   | App u v =>
      match cl_term v eta, c_sk u eta sk with
      | Typ, PROD _ s => s
      | _, s => s
      end
   | _ => PROP
   end.
```

Soundness: `c_sk t eta sk = skeleton T eta` when `T` is a kind and the type of `t`

Classification - Before

```
Lemma class_knd :
  forall (gamma : env) (t T : term),
  typ gamma t T ->
  T = Srt set ->
    cl_term t (class_env gamma)
    = Knd (extract_sk (cl_term t (class_env gamma))).

Lemma class_con :
  forall (gamma : env) (t T : term),
  typ gamma t T ->
  typ gamma T (Srt set) ->
    cl_term t (class_env gamma)
    = Con (extract_c_sk cl_term t (class_env gamma)).

Lemma class_obj :
  forall (gamma : env) (t T : term) (s : sort),
  s = prop ->
  typ gamma t T ->
  typ gamma T (Srt s) ->
    cl_term t (class_env gamma) = Obj.
```

Classification - After

```
Inductive preserves2 : class -> sort -> Prop :=
  | p2_set : preserves2 Con set
  | p2_prop : preserves2 Obj prop.

Definition preserves_term T K eta gamma :=
  K = Srt set /\ cl_term T eta = Knd \/
  (exists s : sort,
    typ gamma K (Srt s) /\ preserves2 (cl_term T eta) s).

Theorem classification gamma T K :
  typ gamma T K ->
  forall eta,
  preserves eta gamma ->
    preserves_term T K eta gamma.
```

Improvements

- Independent use of (constructor-)skeletons and classes
- Requires only a single lemma for all classes
- Much easier to understand (I think)
- Coq agrees: ≈ 600 vs ≈ 1000 lines in that file


B. Barras. "Coq in Coq". 1997.


Herman Geuvers. "A short and flexible proof of Strong Normalization for the Calculus of Constructions". In: *TYPES*. 1994, pp. 14–38.


Z. Luo. *Computation and Reasoning: A Type Theory for Computer Science*. International Series of Monographs on Computer Science. Clarendon Press, 1994. ISBN: 9780198538356. URL: http://books.google.de/books?id=z3uicYzJR1MC.


William W Tait. "Intensional interpretations of functionals of finite type I". In: *The Journal of Symbolic Logic* 32.2 (1967), pp. 198–212.