Formalizing a set-theoretical model of CIC in Coq/IZF

Bruno Barras

INRIA Saclay

May 16, 2011

Overview

- Introduction
  - Models
  - Gödel
  - Coq as an IZF prover
- Method:
  - shallow embedding
  - abstract model
- Calculus of Constructions
  - Consistency model
  - Strong Normalization
- Universes and Grothendieck universes
- Inductive types
  - constructors and pattern-matching
  - type fixpoints and recursive definitions
  - strictly positive inductive types

Motivations

Why a model of CIC ?

- Currently *no model of the full formalism of Coq*:
  features studied separately: Streicher, Coquand, Luo, Werner, H. Goguen
- No *strong intuition* of which axioms are consistent with CIC
  (Chicli-Pottier-Simpson paradox)

Why formally ?

- To be "sure"
- To make it *simpler* (for both the designer and the reader)

Which model do we want ?

- Smallest model vs
  Model with smallest number of assumptions
  (or: studying the proof theoretic strength of CIC vs supporting more axioms)

In particular, we do not limit ourselves to continuous or computable functions (countable model). We want to be able to support classical reals, powerful description axioms, extentionality and what not...

Set-theoretical model:

$$A \to B \text{ set of all set-theoretical function from } A \text{ to } B.$$

Set theory: IZF

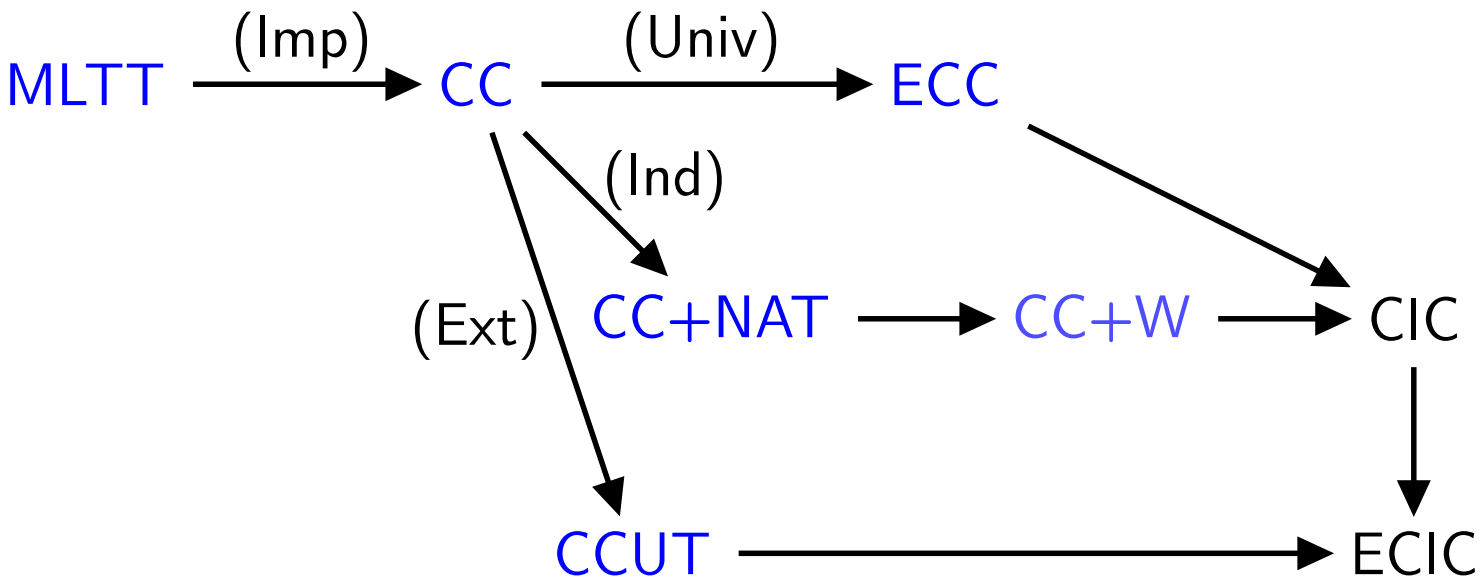
Axiomatized Zermelo-Fraenkel without excluded-middle:

- a carrier type set with equality $=$ and membership $\in$,
- pair $\{a; b\}$,
- union $\bigcup a$,
- powerset $\mathcal{P}(a)$,
- separation $\{x \in A \mid P(x)\}$, (predicate in HOAS)
- replacement $\{y \mid \exists x \in A.R(x, y)\}$ (R functional relation, HOAS),
- infinity
- unused: well-foundation (instead of regularity in ZF)

Library: couples, relations, functions, plump ordinals, fixpoint theorem, Grothendieck universes, ...

Method

The Playground



Three independent features:

- Predicative universes
- Inductive types
- Extentional theory

Semantics first

Usual scheme:

- Introduce the syntax: terms and judgements
- Define the interpretation (recursion over the syntax)
- Prove soundness of the interpretation

Many systems: better avoid to start from the syntax!

- *Shallow embedding*
  (expression constructors and judgements directly build their semantics)
- Naturally *extendable*
  (just define new constructors and derive new rules)
- "Pick" the syntax once done with the semantics

Abstract model

Describes the world of *ground expressions*

Consists of:

- A set $\mathcal{M}$ of denotations (for both objects and types)
- Judgement: $[M : T]$ or $M \in \mathsf{El}(T)$
- Calculus-specific operations and properties (e.g.: $\Lambda, \Pi, @, \beta$)

Two tasks:

- Building an instance of the abstract model
- Dealing with free variables and substitution [routine]

Model construction

Dealing with free variables (de Bruijn):

- constr $\triangleq (\mathbb{N} \to \mathcal{M}) \to \mathcal{M}$ (valuations $\triangleq \mathbb{N} \to \mathcal{M}$)
- substitution $M[0 \backslash N] \triangleq \rho \mapsto M\,(N\rho :: \rho)$
- context $\triangleq$ constr*

Judgements:

- $[\Gamma]$ = set of valid valuations: ρ s.t. $(x : T) \in \Gamma \Rightarrow [x\rho : T\rho]$
- Typing: $[\Gamma \vdash M : T] \triangleq \forall \rho \in [\Gamma], [M\rho : T\rho]$
- Equality: $[\Gamma \vdash M = N] \triangleq \forall \rho \in [\Gamma].\, M\rho = N\rho$

Derive all necessary typing rules (so we have soundness)

$$\Gamma \vdash M : T \Rightarrow [\Gamma \vdash M : T]$$

Calculus of Constructions

Abstract model of Martin Löf's Type Theory

Structure:

- A setoid $(\mathcal{M}, =)$, membership $\_ \in \mathrm{El}(\_)$
- Operations: $\Lambda, @, \Pi$
  - $\Lambda(A, F)$: function F with domain A $\quad A : \mathcal{M} \quad F : \mathcal{M} \to \mathcal{M}$
  - $@(M, N)$: application $\quad M, N : \mathcal{M}$
  - $\Pi(A, B)$: set of dependent functions $\quad A : \mathcal{M} \quad B : \mathcal{M} \to \mathcal{M}$
- Properties:
  - Π-intro: $(\forall x \in El(A).\ F(x) \in El(B(x))) \Rightarrow \Lambda(A, F) \in El(\Pi(A, B))$
    $(\mathrm{cf} \quad x : A \vdash F : B \quad \Rightarrow \quad \vdash \lambda x : A.\ F \ : \ \Pi x : A.\ B)$
  - Π-elim: $M \in El(\Pi(A, B)) \ \wedge \ N \in El(A) \ \Rightarrow \ @(M, N) \in El(B(N))$
    $(\mathrm{cf} \quad \vdash M \ : \ \Pi x : A.\ B \quad \wedge \quad \vdash N : A \quad \Rightarrow \quad \vdash M\ N \ : \ B[x \backslash N])$
  - β-equality: $N \in El(A) \ \Rightarrow \ @(\Lambda(A, F), N) = F(N)$

Straightforward implementation:

- $\mathcal{M}$ = set and El is the identity (alternative: HF)
- Π dependent product (usual encoding of functions)
- Note: Λ *uses* the domain argument

Abstract model of CC

Additional constants and properties:

- Prop: $*$
- Impredicativity: $(\forall x \in \mathrm{El}(A).\ B(x) \in \mathrm{El}(*)) \ \Rightarrow\ \Pi(A, B) \in \mathrm{El}(*)$

Note: topsort Kind is the proper class $\mathcal{M}$:

$$[M : T] \triangleq M \neq \mathrm{Kind} \wedge (T = \mathrm{Kind} \vee M \in \mathrm{El}(T))$$

Implementation:

- Aczel's encoding
$$\begin{array}{rcl} \Lambda(A, F) & \triangleq & \{(x, y) \mid x \in A \wedge y \in F(x)\} \\ @(M, N) & \triangleq & \{y \mid (N, y) \in M\} \end{array}$$
- But this is incompatible with Streicher's method because functions do not carry their domain.

Translation of CC terms towards pure *lambda*-calculus:

- preserving *all* reductions
- preserving strong normalization

Example:

$$\begin{aligned} \lambda x : T.\, M &\mapsto (\lambda xy.x)\ (\lambda x.M)\ T \\ \Pi x : T.\, M &\mapsto (\lambda xy.x)\ (\lambda x.M)\ T \end{aligned}$$

Universes

Inductive types

Overview

Model construction

- Induction: constructors and pattern-matching
- Type with stages
- Recursive functions
- Strict positivity

Judgements

Rule samples

$$(\text{Not: } \Gamma \vdash M : (x:T) \rightsquigarrow U \triangleq \Gamma \vdash M : \Pi x:T.\, U \ \wedge\ \Gamma \vdash M\ (\text{dom}\ T))$$

$$\frac{(\beta < \alpha^+); (x : I^\beta) \vdash U \uparrow \qquad (\beta < \alpha^+); (f : (x : I^\beta) \rightsquigarrow U_{\beta,x}) \vdash M : (x : I^{\beta^+}) \rightsquigarrow U_{\beta^+,x}}{\vdash \mathsf{Fix}(\beta f.M,\, \alpha) : (x : I^\alpha) \rightsquigarrow U_{\alpha,x}}$$

$$\frac{\vdash T \uparrow \qquad (x:T) \vdash_= M : U}{\vdash \lambda x:T.\, M : (x:T) \rightsquigarrow U} \qquad \frac{\vdash M : (x:T) \rightsquigarrow U \qquad \vdash_= N : T}{\vdash_= M\ N : U\{x \backslash N\}}$$

$$\frac{(\beta < \alpha) \in \Gamma}{\Gamma \vdash \beta \uparrow} \qquad \frac{\vdash O \uparrow}{\vdash O^+ \uparrow} \qquad \frac{\vdash O \uparrow}{\vdash I^O \uparrow} \qquad \frac{\vdash T = \qquad (x:T) \vdash U \uparrow}{\vdash \Pi x:T.\, U \uparrow}$$

Expressivity: examples

- Recursor:
  $[\vdash \mathit{Nrec} : \Pi P : \mathsf{nat} \to \mathsf{Prop}.P(0) \to (\Pi k.P(k) \to P(S(k))) \to \Pi n.P(n)]$
- Annotated subtraction: $[\alpha < \infty \vdash \mathsf{minus}_\alpha : \mathsf{nat}^\alpha \to \mathsf{nat} \to \mathsf{nat}^\alpha]$
- Cannot deal with min : $\mathsf{nat}^\alpha \to \mathsf{nat}^\alpha \to \mathsf{nat}^\alpha$

Conclusion