Semantic consistency proofs for systems of illative combinatory logic

Łukasz Czajka

June 1, 2015
Illative combinatory logic

What is it?

▶ The original Combinatory Logic, but fixed.
Illative combinatory logic

What is it?

- The original Combinatory Logic, but fixed.
- Based on a long (but not so popular) line of work initiated by Moses Schönfinkel and Haskell Curry.
Illative combinatory logic

What is it?

- The original Combinatory Logic, but fixed.
- Based on a long (but not so popular) line of work initiated by Moses Schönfinkel and Haskell Curry. (continued by Fitch, Hindley, Seldin, Bunder, Dekkers, Barendregt, . . . )
Illative combinatory logic

What is it?

- The original Combinatory Logic, but fixed.
- Based on a long (but not so popular) line of work initiated by Moses Schönfinkel and Haskell Curry. (continued by Fitch, Hindley, Seldin, Bunder, Dekkers, Barendregt,...)
- A certain approach to logic.
Combinatory logic

The general idea

Reduce systems of logic to a certain “simple” form.
Combinatory logic
The general idea

Reduce systems of logic to a certain “simple” form.

▶ Syntax: only a single binary application operation plus some constants.
Combinatory logic

The general idea

Reduce systems of logic to a certain “simple” form.

- Syntax: only a single binary application operation plus some constants.
- Inference rules: “simple”, not involving “complex” notions like substitution.
Naive combinatory logic
A simple (inconsistent) system

- Terms: applicative terms over constants ⊃, K, S, Q.
Naive combinatory logic
A simple (inconsistent) system

- Terms: applicative terms over constants ⊤, K, S, Q.
- Judgements: \( \Gamma \vdash X \).
Naive combinatory logic
A simple (inconsistent) system

- Terms: applicative terms over constants \( \exists, K, S, Q \).
- Judgements: \( \Gamma \vdash X \).
- Notational conventions:
  - \( X \vdash Y \equiv \exists XY \),
  - \( X = Y \equiv QXY \).
Naive combinatory logic
A simple (inconsistent) system – rules

\[ \Gamma, X \vdash X \]

\[ \frac{\Gamma, X \vdash Y}{\Gamma \vdash X \supset Y} \] \hspace{1cm} \text{(\textit{\supset i})} \hspace{1cm} \frac{\Gamma \vdash X \Gamma \vdash X \supset Y}{\Gamma \vdash Y} \hspace{1cm} \text{(\textit{\supset e})} \]

\[ \Gamma \vdash SXYZ = XZ(YZ) \]

\[ \Gamma \vdash KXY = X \]

\[ \Gamma \vdash X \Gamma \vdash X = Y \]

\[ \Gamma \vdash Y \] \hspace{1cm} \text{(eq)}

+ usual rules for equality.
Naive combinatory logic

Abstraction $\lambda x. X$ with the property

$$\vdash (\lambda x. X) Y = X[Y/x]$$

is definable using combinators.
Abstraction $\lambda x. X$ with the property

$$\vdash (\lambda x. X) Y = X[Y/x]$$

is definable using combinators.

It follows that for every term $X$ with $x$ free there exists a term $M$ such that $\vdash M = X[M/x]$. 

Naive combinatory logic
Naive combinatory logic

Curry's paradox

For an arbitrary given term $Y$, there is $X$ such that

$$\vdash X = (X \supset Y).$$
For an arbitrary given term $Y$, there is $X$ such that
$\vdash X = (X \supset Y)$.

Using $\vdash X = (X \supset Y)$ one shows $\vdash Y$. 
Illative combinatory logic

A simple system

Terms: untyped λ-terms over constants ∀, ⊃, Prop, Type.
Illative combinatory logic
A simple system

- Terms: untyped $\lambda$-terms over constants $\forall, \supset, \text{Prop}, \text{Type}$.
- Notational conventions:
  - $X \supset Y \equiv \supset XY$,
  - $\forall x : X . Y \equiv \forall X (\lambda x . Y)$,
  - $X : Y \equiv YX$. 
Illative combinatory logic

A simple system – rules

\[
\begin{align*}
\Gamma, X & \vdash Y & \Gamma & \vdash X : \text{Prop} & \Rightarrow \quad \Gamma & \vdash X \supset Y \\
\Gamma, Xx & \vdash Yx & \Gamma & \vdash X : \text{Type} & \quad x \notin \text{FV}(\Gamma, X, Y) & \Rightarrow \quad \Gamma & \vdash \forall XY \\
\Gamma & \vdash X & \Gamma & \vdash X \supset Y & \Rightarrow \quad \Gamma & \vdash Y \\
\Gamma & \vdash \forall XY & \quad \Gamma & \vdash XZ & \Rightarrow \quad \Gamma & \vdash YZ
\end{align*}
\]
Illative combinatory logic

A simple system – rules

\[
\frac{\Gamma, X \vdash Y}{\Gamma \vdash X \supset Y} \quad \frac{\Gamma \vdash X \quad \Gamma \vdash X \supset Y}{\Gamma \vdash Y}
\]

\[
\frac{\Gamma, Xx \vdash Yx}{\Gamma \vdash \forall XY} \quad \frac{\Gamma \vdash \forall XY \quad \Gamma \vdash XZ}{\Gamma \vdash YZ}
\]

\[
\frac{\Gamma \vdash X : \text{Prop} \quad \Gamma, X \vdash Y : \text{Prop}}{\Gamma \vdash (X \supset Y) : \text{Prop}}
\]

\[
\frac{\Gamma, Xx \vdash (Yx) : \text{Prop} \quad \Gamma \vdash X : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y)}{\Gamma \vdash (\forall XY) : \text{Prop}}
\]
Illative combinatory logic

A simple system – rules

\[
\begin{align*}
\Gamma, X \vdash Y & \quad \Gamma \vdash X : \text{Prop} \\
\overline{\Gamma \vdash X \supset Y} & \\
\Gamma, X \vdash Y, X & \quad \Gamma \vdash X \supset Y \\
\overline{\Gamma \vdash Y} & \\
\Gamma \vdash X & \quad \Gamma \vdash X \supset Y \\
\overline{\Gamma \vdash Y} & \\
\Gamma, Xx \vdash Yx & \quad \Gamma \vdash X : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y) \\
\overline{\Gamma \vdash \forall XY} & \\
\Gamma \vdash \forall XY & \quad \Gamma \vdash XZ \\
\overline{\Gamma \vdash YZ} & \\
\Gamma \vdash X \vdash X : \text{Prop} & \quad \Gamma, X \vdash Y : \text{Prop} \\
\overline{\Gamma \vdash (X \supset Y) : \text{Prop}} & \\
\Gamma, Xx \vdash (Yx) : \text{Prop} & \quad \Gamma \vdash X : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y) \\
\overline{\Gamma \vdash (\forall XY) : \text{Prop}} & \\
\Gamma \vdash X \quad X =_{\beta \eta} Y & \quad \Gamma \vdash X \vdash X : \text{Prop} \\
\overline{\Gamma \vdash Y} & \\
\Gamma \vdash X & \quad \Gamma \vdash X : \text{Prop} \\
\end{align*}
\]
Induction
For natural numbers

Induction principle **applicable to untyped terms.**

\[
\Gamma \vdash X_0 \quad \Gamma, x : \text{Nat}, Xx \vdash X(sx) \quad x \notin FV(\Gamma, X) \\
\hline
\Gamma \vdash \forall x : \text{Nat}. Xx
\]
Induction
For natural numbers

Induction principle applicable to untyped terms.

\[ \Gamma \vdash X_0 \quad \Gamma, x : \text{Nat}, Xx \vdash X(sx) \quad x \notin FV(\Gamma, X) \]
\[ \Gamma \vdash \forall x : \text{Nat} . Xx \]

The following would be less useful:

\[ \forall f : \text{Nat} \rightarrow \text{Prop} . ((f0 \land (\forall x : \text{Nat} . fx \supset f(sx))) \supset \forall x : \text{Nat} . fx) \]
“Types”
Notational conventions

\[ X \rightarrow Y \equiv \lambda f. \forall x : X. (fx : Y), \]
“Types”

Notational conventions

- \( X \rightarrow Y \equiv \lambda f. \forall x : X. (fx : Y) \),
- \( (x : X) \rightarrow Y(x) \equiv \lambda f. \forall x : X. (fx : Y(x)) \),
“Types”
Notational conventions

- \( X \rightarrow Y \equiv \lambda f. \forall x : X. (fx : Y) \),
- \((x : X) \rightarrow Y(x) \equiv \lambda f. \forall x : X. (fx : Y(x))\),
- \(\Sigma (x : X) Y(x) \equiv \lambda p. (\pi_1 p : X) \land (\pi_2 p : Y(\pi_1 p))\),
“Types”
Notational conventions

- $X \to Y \equiv \lambda f. \forall x : X. (f x : Y)$,
- $(x : X) \to Y(x) \equiv \lambda f. \forall x : X. (f x : Y(x))$,
- $\Sigma(x : X) Y(x) \equiv \lambda p. (\pi_1 p : X) \land (\pi_2 p : Y(\pi_1 p))$,
“Types”
Postulated formation rules

\[
\Gamma \vdash X : \text{Type} \quad \Gamma, x : X \vdash Y : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y) \\
\Gamma \vdash (X \to Y) : \text{Type}
\]

\[
\Gamma \vdash X : \text{Type} \quad \Gamma, x : X \vdash Y(x) : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y) \\
\Gamma \vdash ((x : X) \to Y(x)) : \text{Type}
\]

\[
\Gamma \vdash X : \text{Type} \quad \Gamma, x : X \vdash Y(x) : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y) \\
\Gamma \vdash (\Sigma(x : X) Y(x)) : \text{Type}
\]

\[
\ldots
\]
"Types"

Derived rules

Γ, x : X ⊢ Z : Y  Γ ⊢ X : Type  \( x \notin \text{FV}(\Gamma, X, Y) \)
Γ ⊢ (\lambda x.Z) : X → Y

Γ ⊢ F : X → Y  Γ ⊢ Z : X
Γ ⊢ FZ : Y
“Types”

Derived rules

\[
\Gamma, x : X \vdash Z : Y(x) \quad \Gamma \vdash X : \text{Type} \quad x \notin \text{FV}(\Gamma, X, Y)
\]

\[
\Gamma \vdash (\lambda x. Z) : (x : X) \to Y(x)
\]

\[
\Gamma \vdash F : (x : X) \to Y(x) \quad \Gamma \vdash Z : X
\]

\[
\Gamma \vdash FZ : YZ
\]
Using the induction principle(s) one may show that simply-typable (dependently-typable, . . . ?) terms extented with well-founded recursion are still “typable” in our system.
⊢ ∀x, y : TermCode. checktype(x, y) ⊃ (eval(x) : eval(y))
Type checking/inference algorithms as proof tactics.
Some loose connections

- NuPRL.
- PX: A computational logic.
- Logical frameworks.
Technical results

1. Conservativity of a classical higher-order illative system over standard semantics for higher-order logic.

2. Consistency of classical higher-order illative systems with dependent types, predicate subtypes, W-types, and:
   - Non-constructive choice.
   - Conditional (branching on truth-values).
   - Extensionality for W-types.
Technical results

Model construction

A model is a tuple \( \langle \mathcal{C}, \mathcal{T}, \mathcal{F}, \bot, v, \forall, \ldots \rangle \) where:

1. \( \bot \in \mathcal{F} \),
2. \( \mathcal{T} \cap \mathcal{F} = \emptyset \).
3. \( v \cdot a \cdot b \in \mathcal{T} \) iff \( a \in \mathcal{T} \) or \( b \in \mathcal{T} \),
4. \( v \cdot a \cdot b \in \mathcal{F} \) iff \( a \in \mathcal{F} \) and \( b \in \mathcal{F} \),
5. \( \forall \cdot a \cdot b \in \mathcal{T} \) iff \( \text{Type} \cdot a \in \mathcal{T} \) and for every \( c \in \mathcal{C} \) with \( a \cdot c \in \mathcal{C} \) we have \( b \cdot c \in \mathcal{C} \),
6. \( \forall \cdot a \cdot b \in \mathcal{F} \) iff \( \text{Type} \cdot a \in \mathcal{T} \) and there exists \( c \in \mathcal{C} \) with \( a \cdot c \in \mathcal{T} \) and \( b \cdot c \in \mathcal{F} \),
7. if \( \text{Type} \cdot a \in \mathcal{T} \) and \( \text{Type} \cdot b \in \mathcal{T} \), then \( \text{Type} \cdot (\rightarrow \cdot a \cdot b) \in \mathcal{T} \),
8. \( \ldots \)
Technical results

Model construction

We construct a term model parameterised by a standard model $\mathcal{N}$ for higher-order logic.

$$\mathcal{N} = \langle \{ \mathcal{D}_\tau \mid \tau \in \mathcal{T} \}, I \rangle$$
We construct a term model parameterised by a standard model $\mathcal{N}$ for higher-order logic.

$$\mathcal{N} = \langle \{ D_\tau \mid \tau \in \mathcal{T} \}, I \rangle$$

Types of higher-order logic:

$$\mathcal{T} ::= o \mid i \mid \mathcal{T} \rightarrow \mathcal{T}$$
Technical results
Model construction

We construct a term model parameterised by a standard model $\mathcal{N}$ for higher-order logic.

$$\mathcal{N} = \langle \{ D_\tau \mid \tau \in \mathcal{T} \}, I \rangle$$

Types of higher-order logic:

$$\mathcal{T} ::= o \mid i \mid \mathcal{T} \to \mathcal{T}$$

For a type $\tau \in \mathcal{T}$ and an ordinal $\alpha$ we define the representation relations $\succ^\alpha_\tau \in \mathcal{T} \times \mathcal{T}$, the contraction relation $\to^\alpha \in \mathcal{T} \times \mathcal{T}$, and the relation $\succ^\alpha_{\mathcal{G}} \in \mathcal{T} \times \mathcal{G}$ inductively.
Technical results

Model construction

\((\beta)\) \((\lambda x. X) Y \rightarrow^\alpha X[x/Y],\)

\((\gamma)\) \(fX \rightarrow^\alpha b\) if \(f \in D_{\tau_1 \rightarrow \tau_2}, a \in D_{\tau_1}, b \in D_{\tau_2}, f^N(a) = b\) and \(X \succ^<_\tau a,\)

\((\mathcal{F}_{\tau})\) \(X \succ^\alpha_{\tau} d\) if \(\tau = \tau_1 \rightarrow \tau_2, d \in D_{\tau_1 \rightarrow \tau_2}\) and for every \(a \in D_{\tau_1}\) we have \(Xa \sim^<_{\tau_2} d^N(a),\)

\((V_\top)\) \(X \lor Y \succ^\alpha_\top \) if \(X \succ^<_\top \) or \(Y \succ^<_\top,\)

\((V_\bot)\) \(X \lor Y \succ^\alpha_\bot\) if \(X \succ^<_\bot\) and \(Y \succ^<_\bot,\)

\((\forall_\top)\) \(\forall XY \succ^\alpha_\top\) if \(X \succ^<_\top \tau\) and for every \(d \in D_\tau\) we have \(Yd \sim^<_\top \)

\((\forall_\bot)\) \(\forall XY \succ^\alpha_\bot\) if \(X \succ^<_\bot \tau\) and there exists \(d \in D_\tau\) with \(Yd \sim^<_\bot \)

\((F_{\mathcal{F}})\) \(X \rightarrow Y \succ^\alpha_{\mathcal{F}} \tau_1 \rightarrow \tau_2\) if \(X \succ^<_\mathcal{F} \tau_1\) and \(Y \succ^<_\mathcal{F} \tau_2,\)

\(\ldots\)
Technical results

Difficulty

Elements of type $\text{Prop} \rightarrow \tau$, $(\alpha \rightarrow \text{Prop}) \rightarrow \tau$, etc. for $\tau \neq \text{Prop}$. 