The K axiom in Coq (almost) for free[1]

Pierre Corbineau

Université Joseph Fourier – Grenoble 1
Laboratoire Verimag

Réunion ADT Coq
La Ciotat, 2nd February 2010

⇒

[1] Work realized at Radboud Universiteit Nijmegen

Introduction

An implementation of K in Coq

Other useful instances

Equiconsistency of CIC + $\mathcal{C}^+$ with CIC + K + κ

Conclusion & Further work

The Altenkirch-Streicher K axiom

Using Coq notations :

- Identity relation
  eq : $\forall A : \text{Type}, A \to A \to \text{Prop}$ (notation $x =_A y$)
  refl : $\forall A : \text{Type}, \forall x : A, x =_A x$

The Altenkirch-Streicher K axiom

Using Coq notations :

- Identity relation
  eq : $\forall A : \text{Type}, A \rightarrow A \rightarrow \text{Prop}$ (notation $x =_A y$)
  refl : $\forall A : \text{Type}, \forall x : A, x =_A x$
- Uniqueness of Identity Proofs (UIP) :
  $\forall A : \text{Type}, \forall x, y : A, \forall p, q : x =_A y, p =_{x =_A y} q$

The Altenkirch-Streicher K axiom

Using Coq notations :

- Identity relation
  eq : $\forall A : \text{Type}, A \rightarrow A \rightarrow \text{Prop}$ (notation $x =_A y$)
  refl : $\forall A : \text{Type}, \forall x : A, x =_A x$
- Uniqueness of Identity Proofs (UIP) :
  $\forall A : \text{Type}, \forall x, y : A, \forall p, q : x =_A y, p =_{x =_A y} q$
- The Altenkirch-Streicher K axiom :
  K : $\forall A : \text{Type}, \forall x : A, \forall P : x =_A x \rightarrow \text{Type}, P\,(refl_A\, x) \rightarrow \forall h : x =_A x, P\, h$
  reduction : $K\, A\, x\, P\, h\, (refl_A\, x) \rightsquigarrow_\kappa h$

Historical background

- Per Martin-Löf (1973,1984)
  Intuitionistic Type Theory
  Proofs as first class objects
  Allows to express UIP and K, not to derive them

Historical background

- Per Martin-Löf (1973,1984)
  Intuitionistic Type Theory
  Proofs as first class objects
  Allows to express UIP and K, not to derive them

- Thomas Streicher & Martin Hoffman (1994) :
  The groupoid interpretation of Type Theory
  ITT $\nvdash$ K (Counter-model)

Historical background

- Per Martin-Löf (1973,1984)
  Intuitionistic Type Theory
  Proofs as first class objects
  Allows to express UIP and K, not to derive them
- Thierry Coquand (1992)
  Pattern-matching with dependent types
  ALF implements the K axiom :
  $\{e \mapsto refl_A\, x\}$ covering for $\{A : Set, x : A, e : x =_A x\}$
  hence $K\, A\, x\, P\, h\, (refl_A\, x) \mapsto h$ yields a valid definition
- Thomas Streicher & Martin Hoffman (1994) :
  The groupoid interpretation of Type Theory
  ITT $\not\vdash$ K (Counter-model)

Problematics

In the Coq system :

- K is independent from CIC
- K doesn't fit the general scheme for inductive families
  - eq : $\forall A$: Type, $A \rightarrow A \rightarrow$ Prop (notation $x =_A y$)
  - elimination predicate must be $P : \forall y : A, x =_A y \rightarrow$ Type
- K : eliminator for subfamily $\lambda A.\lambda x.x =_A x$
  We need $P : x =_A x \rightarrow$ Type

Solution : relax the pattern-matching typing rule

Expected outcome

Internalisation of K into the Coq syntax

Simple syntactic counterparts for well-known consequences of K

- Equivalence between Heterogeneous (JMeq) and Leibniz equalities
- Monomorphic replacement for JMeq
- Injectivity of the second projection for dependent pairs
- Equality between deBruijn telescopes

Major consequence : JMeq as the default equality

- Equational reasoning with dependent types

Introduction

An implementation of K in Coq

Other useful instances

Equiconsistency of CIC + $\mathcal{C}^+$ with CIC + K + κ

Conclusion & Further work

Changing replacement into K

- Leibniz equality
  eq : $\forall A : \text{Type}, A \to A \to \text{Prop}$ (notation $x =_A y$)
  refl : $\forall A : \text{Type}, \forall x : A, x =_A x$
- Dependent elimination scheme

$$\begin{array}{l}
\lambda A : \text{Type}\,.\lambda x : A. \\
\lambda P : \quad \forall y : A, \quad x =_A y \to \text{Type}\,. \\
\lambda H : P\,x\,(refl_A\,x). \\
\qquad \lambda y : A. \quad \lambda h : x =_A y. \\
\quad \text{match}\ h\ \text{in}\ _ = y \qquad\qquad \text{return}\ P\ y\ h \\
\quad \text{with}\ refl \Rightarrow H\ \text{end}
\end{array}$$

We force the value of indices with let . . . in definitions.

Changing replacement into K

- Leibniz equality
  eq : $\forall A : \text{Type}, A \to A \to \text{Prop}$ (notation $x =_A y$)
  refl : $\forall A : \text{Type}, \forall x : A, x =_A x$
- K axiom

$$
\begin{array}{l}
\lambda A : \text{Type}.\lambda x : A.\\
\lambda P : \qquad\qquad x =_A x \to \text{Type}.\\
\lambda H : P \quad (refl_A\, x).\\
\qquad\qquad \lambda h : x =_A x.\\
\quad \text{match } h \text{ in } \_ = x \qquad\qquad \text{return } P \quad h\\
\quad \text{with } refl \Rightarrow H \text{ end}
\end{array}
$$

We force the value of indices with let . . . in definitions.

Changing replacement into K

- Leibniz equality
  $\text{eq} : \forall A : \text{Type}, A \rightarrow A \rightarrow \text{Prop}$ (notation $x =_A y$)
  $\text{refl} : \forall A : \text{Type}, \forall x : A, x =_A x$
- K axiom with let . . . in

$$\lambda A : \text{Type} . \lambda x : A.$$
$$\lambda P : \text{let } y : A := x \text{ in } x =_A y \rightarrow \text{Type} .$$
$$\lambda H : P \quad (refl_A\, x).$$
$$\text{let } y : A := x \text{ in } \lambda h : x =_A y.$$
$$\quad \text{match } h \text{ in } \_ = y \text{ where } y := x \text{ return } P\, y\, h$$
$$\quad \text{with } refl \Rightarrow H \text{ end}$$

We force the value of indices with let . . . in definitions.

A generic inductive definition

Example

Inductive I $(p_1 : P_1)\ldots(p_{n'} : P_m) : \forall(y_1 : A_1)\ldots(y_p : A_{n''}), \mathcal{S}$:=

$C_1 : \forall(b_{1,1} : B_{1,1})\ldots(b_{1,k_1} : B_{1,k_1}), I\ p_1\ldots p_m\ t_{1,1}\ldots t_{1,p}$

$\ldots$

$\mid C_n : \forall(b_{n,1} : B_{n,1})\ldots(b_{n,k_n} : B_{n,k_n}), I\ p_1\ldots p_m\ t_{n,1}\ldots t_{n,p}$.

Nomenclature :

- $p_1\ldots p_{n'}$ parameters
- $y_1\ldots y_{n''}$ indices
- $b_{i,1}\ldots b_{i,k_i}$ arguments of constructor C_i

The current typing rule

Typing rule for pattern-matching on I

$$\frac{\Gamma \vdash u : I\,\vec{p}\,\vec{a} \quad \Gamma, \vec{y} : \vec{A}, x : (I\,\vec{p}\,\vec{y}) \vdash P : \mathcal{S}' \qquad \Gamma, \vec{b_i} : \vec{B_i} \vdash F_i : P[\vec{t_i}/\vec{y}; (C_i^I\,\vec{p}\,\vec{b_i})/x] \quad {}_{i=1..n}}{\Gamma \vdash \begin{array}{l}\textbf{match}\ u\ \textbf{as}\ x\ \textbf{in}\ I\ y_1 \ldots y_{n''}\ \textbf{return}\ P\ \textbf{with} \\ \quad C_1\,b_{1,1} \ldots b_{1,k_1} \Rightarrow F_1 \\ \quad \vdots \\ \mid C_1\,b_{n,1} \ldots b_{n,k_n} \Rightarrow F_n \\ \textbf{end}\end{array} : P[\vec{a}/\vec{y}; u/x]} \mathcal{C}$$

The new typing rule

Instantiation of contexts :
declaration $(y : T)$ in Γ $(y \in dom\,\sigma)$ becomes
local definition let $y := \sigma y$ in in $\Gamma\sigma$

$$\frac{\begin{array}{l}\Gamma \vdash u : I\,\vec{p}\,\vec{a} \quad a_j \approx y_j\sigma_{(y_j \in dom\,\sigma)} \quad \Gamma, (\vec{y} : \vec{A})\sigma, x : (I\,\vec{p}\,\vec{y}) \vdash P : \mathcal{S}' \\ \Gamma, \vec{b_i} : \vec{B_i} \vdash F_i : (P\sigma)[\vec{t_i}/\vec{y}; (C_i^I\,\vec{p}\,\vec{b_i})/x] \quad t_j \approx y_j\sigma_{(y_j \in dom\,\sigma)} \quad i=1..n\end{array}}{\Gamma \vdash \begin{array}{l}\textbf{match}\ u\ \textbf{as}\ x\ \textbf{in}\ I\ y_1 \ldots y_{n''}\ \textbf{where}\ \sigma\ \textbf{return}\ P\ \textbf{with} \\ \quad C_1\,b_{1,1} \ldots b_{1,k_1} \Rightarrow F_1 \\ \quad \vdots \\ \mid C_1\,b_{n,1} \ldots b_{n,k_n} \Rightarrow F_n \\ \textbf{end}\end{array} : P\sigma[\vec{a}/\vec{y}; u/x]}\ \mathcal{C}^+$$

The new reduction rule

Same ι-rule as before.

Introduction

An implementation of K in Coq

Other useful instances

Equiconsistency of CIC + $\mathcal{C}^+$ with CIC + K + κ

Conclusion & Further work

The (J.M.) heterogeneous equality

- Definition
  $\mathrm{JMeq} : \forall A : \mathsf{Type}, A \rightarrow \forall B : \mathsf{Type}\ B \rightarrow \mathsf{Prop}$ (notation $(x : A) = (y : B)$)
  $\mathrm{JMrefl} : \forall A : \mathsf{Type}, \forall x : A, (x : A) =_A (x : A)$
- Elimination scheme : Polymorphic replacement
  $\lambda A : \mathsf{Type}.\lambda x : A.$
  $\lambda P : \quad \forall B : \mathsf{Type}, \quad \forall y : B,$
  $(x : A) = (y : B) \rightarrow \mathsf{Type}.$
  $\lambda H : P\,A\,x\,(JMrefl_A\,x).$
  $\lambda B : \mathsf{Type}, \quad \lambda y : B, \quad \lambda h : (x : A) = (y : B).$
  $\mathsf{match}\ h\ \mathsf{in}(\_ : \_) = (y : B)$
  $\mathsf{return}\ P\,A\,x\,h$
  $\mathsf{with}\ JMrefl \Rightarrow H\ \mathsf{end}$

The (J.M.) heterogeneous equality

- Definition
  JMeq : $\forall A : \text{Type}, A \to \forall B : \text{Type}\ B \to \text{Prop}$ (notation $(x : A) = (y : B)$)
  JMrefl : $\forall A : \text{Type}, \forall x : A, (x : A) =_A (x : A)$
- Elimination scheme : Monomorphic replacement
  $\lambda A : \text{Type} . \lambda x : A.$
  $\lambda P :$ let B : Type := A in $\forall y : B,$
  $(x : A) = (y : B) \to \text{Type}\,.$
  $\lambda H : P \quad x\,(JMrefl_A\, x).$
  let B : Type := A in $\lambda y : B, \quad \lambda h : (x : A) = (y : B).$
  match h in $(\_ : \_) = (y : B)$ where $B := A$
  return $P \quad x\ h$
  with $JMrefl \Rightarrow H$ end

The (J.M.) heterogeneous equality

- Definition
  JMeq : $\forall A : \text{Type}, A \to \forall B : \text{Type}\ B \to \text{Prop}$ (notation $(x : A) = (y : B)$)
  JMrefl : $\forall A : \text{Type}, \forall x : A, (x : A) =_A (x : A)$
- Elimination scheme : K axiom for JMeq
  $\lambda A : \text{Type} . \lambda x : A.$
  $\lambda P :$ let $B : \text{Type} := A$ in let $y : B := x$ in
  $(x : A) = (y : B) \to \text{Type}.$
  $\lambda H : P \quad (JMrefl_A\, x).$
  let B : Type $:= A$ in let $y : B := x$ in $\lambda h : (x : A) = (y : B).$
  match h in $(\_ : \_) = (y : B)$ where $B := A; y := x$
  return $P \quad h$
  with $JMrefl \Rightarrow H$ end

Injectivity of second projection

We want to show that

$$\forall (P : A \to \textcolor{blue}{\mathsf{Type}})(a : A)(x, y : P\,a), \\ \langle a, x \rangle =_{\Sigma P} \langle a, y \rangle \to x =_{(P a)} y$$

First we show that

$$\forall (P : A \to \textcolor{blue}{\mathsf{Type}})(a : A)(x, y : P\,a), \\ \langle a, x \rangle =_{\Sigma P} \langle a, y \rangle \to (x : P\,a) = (y : P\,a)$$

by elimination along $\lambda p : \Sigma P.(x : P\,a) = (\pi_2\,p : P\,(\pi_1\,p))$ We conclude using the new JM monomorphic replacement scheme.

Introduction

An implementation of K in Coq

Other useful instances

Equiconsistency of CIC + $\mathcal{C}^+$ with CIC + K + κ

Conclusion & Further work

Proof Sketch

We show the equiconsistency of CIC + $\mathcal{C}^+$ and CIC + K + κ by making each system simulate the reductions of each other.

1. K and κ can be simulated using $\mathcal{C}^+$
2. We need a simulation of CIC + $\mathcal{C}^+$ in CIC + K + κ
   Idea : encode inductive families using equality
   Works only for non-recursive families

Translation of Inductive families

Indices of non-recursive family I become parameters of family I'.
The inductive set I' has constructors $C_i^{I'}, i = 1..n$ such that :

$$C_i^{I'}\ \vec{p}\ \vec{a} : \forall \vec{b_i} : \vec{B_i}.\langle \vec{t_i}\ \vec{b_i} \rangle = \langle \vec{a} \rangle \to I'\ \vec{p}\ \vec{a}$$

We can then translate

$$\widehat{C_i^I} =_{\text{def}} \lambda \vec{p}.\lambda \vec{b_i}.C_i^{I'}\ \vec{p}\ (\widehat{\vec{t_i}}\ \vec{b_i})\ \vec{b_i}\ (\text{refl}_=\ \langle \widehat{\vec{t_i}}\ \vec{b_i} \rangle)$$

Translation of pattern-matching

A match expression like :

$$\textbf{match}\ u\ \textbf{as}\ x\ \textbf{in}\ I\ y_1 \ldots y_{n''}\ \textbf{where}\ \sigma\ \textbf{return}\ P\ \textbf{with}$$
$$C_1\,\vec{b_1} \Rightarrow F_1$$
$$\vdots$$
$$|\ C_1\,\vec{b_n} \Rightarrow F_n$$
$$\textbf{end}$$

becomes

$$\textbf{match}\ u\ \textbf{as}\ x\ \textbf{in}\ I'\ \textbf{return}\ P\sigma[\vec{a}/\vec{y}; u/x]\ \textbf{with}$$
$$C_1\,\vec{b_1}\,\vec{e}_1 \Rightarrow \{\vec{J}/\vec{K}\}F_1\,\vec{e_1}$$
$$\vdots$$
$$|\ C_1\,\vec{b_n}\vec{e}_n \Rightarrow \{\vec{J}/\vec{K}\}F_n\vec{e_n}$$
$$\textbf{end}$$

Use of J or K depends on whether $y \in dom\,\sigma$ or not.

Introduction

An implementation of K in Coq

Other useful instances

Equiconsistency of CIC + $\mathcal{C}^+$ with CIC + K + κ

Conclusion & Further work

Implementation ?

- No conceptual difficulty (only conversion checks)
- Engineering problem : keeping track of σ
  - as non-removable let-in's
  - as a telescope
  - as a list of λ-abstractions
- Parallel proposals :
  - Full proof-irrelevance (B. Werner)
  - K + inversion constraints (J.-L. Sacchini et al.)
- Lots of implementation ahead (kernel, virtual machine)
- Porting Coq equality to JMeq might have unforeseen consequences