A Mechanized Translation from Higher-Order Logic to Set Theory

Alexander Krauss and Andreas Schropp

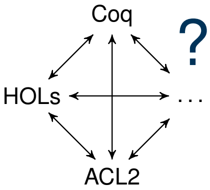
Theorem Proving Group
Technische Universität München

ITP-10, Edinburgh, 2010

Motivation (Long Term)

Explore set theory for interactive theorem proving:

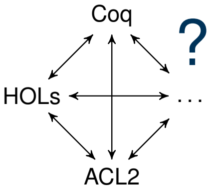
1. Exchange format between proof assistants?
   (“Grand Challenge”, Gordon ’08)

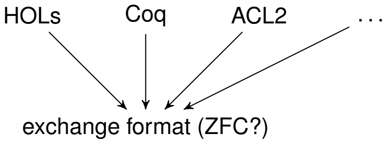


Motivation (Long Term)

Explore set theory for interactive theorem proving:

1. Exchange format between proof assistants?
   (“Grand Challenge”, Gordon ’08)

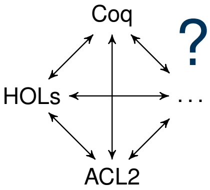


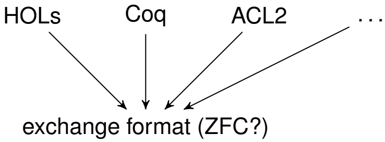


Motivation (Long Term)

Explore set theory for interactive theorem proving:

1. Exchange format between proof assistants?
   ("Grand Challenge", Gordon '08)





2. Soft Types?
   - Flexible type checking on top of an untyped logic
   - Escape HOL's limitations without buying into dependent type theories

Motivation (Short Term)

Explore HOL-style reasoning on top of ZF

- Standard HOL
- Isabelle-specific extensions:
  - Type classes
  - Overloading

Motivation (Short Term)

Explore HOL-style reasoning on top of ZF

- Standard HOL
- Isabelle-specific extensions:
  - Type classes
  - Overloading

Translate actual theories:

- Make the large Isabelle/HOL developments available in ZF
- Facilitate building proof tools

Outline

The Framework: Isabelle/Pure

(aka λ-HOL)

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

Terms

Proofs

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ	$::=$	α	type variable
	$\mid$	$\kappa\ \overline{\tau_n}$	type constructor

Terms

Proofs

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ ::=	α	type variable	*Special types:*
|	$\kappa\ \overline{\tau_n}$	type constructor	*prop*, $\Rightarrow$

Terms

Proofs

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ	$::=$	α	type variable	*Special types:*
	$\mid$	$\kappa\ \overline{\tau_n}$	type constructor	$prop, \Rightarrow$

Terms

t	$::=$	x	variable
	$\mid$	$t_1\ t_2$	application
	$\mid$	$\lambda\, x : \tau.\ t$	abstraction
	$\mid$	$c[\overline{\tau_n}]$	constant

Proofs

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ	$::=$	α	type variable	*Special types:*
	$\mid$	$\kappa\ \overline{\tau_n}$	type constructor	*prop*, $\Rightarrow$

Terms

t	$::=$	x	variable	*Special constants:*
	$\mid$	$t_1\ t_2$	application	$\Longrightarrow$: *prop* $\Rightarrow$ *prop* $\Rightarrow$ *prop*
	$\mid$	$\lambda x : \tau.\ t$	abstraction	$\bigwedge$: $(\alpha \Rightarrow$ *prop*$) \Rightarrow$ *prop*
	$\mid$	$c[\overline{\tau_n}]$	constant	

Proofs

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ	$::=$	α	type variable	*Special types:*
	$\mid$	$\kappa\ \overline{\tau_n}$	type constructor	*prop*, $\Rightarrow$

Terms

t	$::=$	x	variable	*Special constants:*
	$\mid$	$t_1\ t_2$	application	$\Longrightarrow : prop \Rightarrow prop \Rightarrow prop$
	$\mid$	$\lambda\, x : \tau.\ t$	abstraction	$\bigwedge : (\alpha \Rightarrow prop) \Rightarrow prop$
	$\mid$	$c[\overline{\tau_n}]$	constant	

Proofs

p	$::=$	h	proof variable (hypothesis)
	$\mid$	$\lambda\, x : \tau.\ p$	abstraction over terms
	$\mid$	$\lambda\, h : \phi.\ p$	abstraction over proofs
	$\mid$	$p \odot t$	application of terms
	$\mid$	$p_1 \bullet p_2$	application of proofs
	$\mid$	$thm[\overline{\tau_n}]$	proof constant (theorem/axiom)

The Framework: Isabelle/Pure

(aka λ-HOL)

Types

τ	$::=$	α	type variable	*Special types:*
	$\mid$	$\kappa\ \overline{\tau_n}$	type constructor	$prop, \Rightarrow$

Terms

t	$::=$	x	variable	*Special constants:*
	$\mid$	$t_1\ t_2$	application	$\Longrightarrow\ : prop \Rightarrow prop \Rightarrow prop$
	$\mid$	$\lambda x : \tau.\ t$	abstraction	$\bigwedge\ : (\alpha \Rightarrow prop) \Rightarrow prop$
	$\mid$	$c[\overline{\tau_n}]$	constant	

Proofs

p	$::=$	h	proof variable (hypothesis)
	$\mid$	$\lambda x : \tau.\ p$	abstraction over terms $\bigwedge I$
	$\mid$	$\lambda h : \phi.\ p$	abstraction over proofs $\Longrightarrow I$
	$\mid$	$p \odot t$	application of terms $\bigwedge E$
	$\mid$	$p_1 \bullet p_2$	application of proofs $\Longrightarrow E$
	$\mid$	$thm[\overline{\tau_n}]$	proof constant (theorem/axiom)

HOL

types $\mathit{bool}, \ldots$

constants

$$
\begin{array}{lcl}
[\cdot] & : & \mathit{bool} \Rightarrow \mathit{prop} \\
\forall & : & (\alpha \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool} \\
\longrightarrow, \vee, \wedge & : & \mathit{bool} \Rightarrow \mathit{bool} \Rightarrow \mathit{bool} \\
= & : & \alpha \Rightarrow \alpha \Rightarrow \mathit{bool} \\
\mathit{THE} & : & (\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha \\
\mathit{undefined} & : & \alpha
\end{array}
$$

axioms

$$\bigwedge P : \alpha \Rightarrow \mathit{bool}.\ (\bigwedge x : \alpha.\ [\,P\ x\,]) \Longrightarrow [\,\forall x.\ P\ x\,]$$

$$\bigwedge (P : \alpha \Rightarrow \mathit{bool})\ (a : \alpha).\ [\,\forall x.\ P\ x\,] \Longrightarrow [\,P\ a\,]$$

$$\bigwedge P\ Q : \mathit{bool}.\ ([\,P\,] \Longrightarrow [\,Q\,]) \Longrightarrow [\,P \longrightarrow Q\,]$$

$$\bigwedge P\ Q : \mathit{bool}.\ [\,P \longrightarrow Q\,] \Longrightarrow [\,P\,] \Longrightarrow [\,Q\,]$$

$$\bigwedge P : \mathit{bool}.\ [\,P = \mathit{True} \vee P = \mathit{False}\,]$$

$$\vdots$$

HOL

types $\mathit{bool}, \ldots$

constants

$$
\begin{array}{lcl}
[\,\cdot\,] & : & \mathit{bool} \Rightarrow \mathit{prop} \\
\forall & : & (\alpha \Rightarrow \mathit{bool}) \Rightarrow \mathit{bool} \\
\longrightarrow, \vee, \wedge & : & \mathit{bool} \Rightarrow \mathit{bool} \Rightarrow \mathit{bool} \\
= & : & \alpha \Rightarrow \alpha \Rightarrow \mathit{bool} \\
\mathit{THE} & : & (\alpha \Rightarrow \mathit{bool}) \Rightarrow \alpha \\
\mathit{undefined} & : & \alpha
\end{array}
$$

axioms

$$\bigwedge P : \alpha \Rightarrow \mathit{bool}.\ (\bigwedge x : \alpha.\ [\,P\ x\,]) \Longrightarrow [\,\forall x.\ P\ x\,]$$

$$\bigwedge (P : \alpha \Rightarrow \mathit{bool})\ (a : \alpha).\ [\,\forall x.\ P\ x\,] \Longrightarrow [\,P\ a\,]$$

$$\bigwedge P\ Q : \mathit{bool}.\ ([\,P\,] \Longrightarrow [\,Q\,]) \Longrightarrow [\,P \longrightarrow Q\,]$$

$$\bigwedge P\ Q : \mathit{bool}.\ [\,P \longrightarrow Q\,] \Longrightarrow [\,P\,] \Longrightarrow [\,Q\,]$$

$$\bigwedge P : \mathit{bool}.\ [\,P = \mathit{True} \vee P = \mathit{False}\,]$$

$$\vdots$$

ZF

types o, ι

constants

$$
\begin{array}{lcl}
[\,\cdot\,] & : & o \Rightarrow \mathit{prop} \\
\forall & : & (\iota \Rightarrow o) \Rightarrow o \\
\longrightarrow, \vee, \wedge & : & o \Rightarrow o \Rightarrow o \\
= & : & \iota \Rightarrow \iota \Rightarrow o \\
\in & : & \iota \Rightarrow \iota \Rightarrow o
\end{array}
$$

axioms

FOL rules + ZFC axioms

definable

$$
\begin{array}{lcl}
(\lambda \cdot \in \cdot .\ \cdot) & : & \iota \Rightarrow (\iota \Rightarrow \iota) \Rightarrow \iota \\
` & : & \iota \Rightarrow \iota \Rightarrow \iota \\
[\langle \cdot \rangle] & : & \iota \Rightarrow \mathit{prop}
\end{array}
$$

Translation Scheme

Type τ

Set $[\![\tau]\!] : \iota$

Translation Scheme

Type τ	Set $[\![\, \tau \,]\!] : \iota$
Type constructor	Operation on sets

Translation Scheme

Type τ	Set $[\![\tau]\!] : \iota$
Type constructor	Operation on sets
Term $t : \tau$	Term $[\![t]\!] : \iota$ (s.t. $[\![t]\!] \in [\![\tau]\!]$)

Translation Scheme

Type τ	Set $[\![\tau]\!] : \iota$
Type constructor	Operation on sets
Term $t : \tau$	Term $[\![t]\!] : \iota$ (s.t. $[\![t]\!] \in [\![\tau]\!]$)
$(\lambda x : \tau. \ \ldots)$ and $t_1 \ t_2$	$(\lambda\, x \in [\![\tau]\!]. \ \ldots)$ and $t_1 ` t_2$

Translation Scheme

Type τ	Set $[\![\, \tau \,]\!] : \iota$
Type constructor	Operation on sets
Term $t : \tau$	Term $[\![\, t \,]\!] : \iota$ (s.t. $[\![\, t \,]\!] \in [\![\, \tau \,]\!]$)
$(\lambda x : \tau.\ \ldots)$ and $t_1\ t_2$	$(\lambda\, x \in [\![\, \tau \,]\!].\ \ldots)$ and $t_1 ` t_2$
Polymorphic types	$\bigwedge$-quantification over sets

Translation Scheme

Type τ	Set $[\![\,\tau\,]\!] : \iota$
Type constructor	Operation on sets
Term $t : \tau$	Term $[\![\,t\,]\!] : \iota$ (s.t. $[\![\,t\,]\!] \in [\![\,\tau\,]\!]$)
$(\lambda x : \tau.\ \ldots)$ and $t_1\ t_2$	$(\lambda\, x \in [\![\,\tau\,]\!].\ \ldots)$ and $t_1 ` t_2$
Polymorphic types	$\bigwedge$-quantification over sets
Types of variables $v : \tau$	Assumptions $[\![\,v\,]\!] \in [\![\,\tau\,]\!] \implies \ldots$

Translation Scheme

Type τ	Set $[\![\tau]\!] : \iota$
Type constructor	Operation on sets
Term $t : \tau$	Term $[\![t]\!] : \iota$ (s.t. $[\![t]\!] \in [\![\tau]\!]$)
$(\lambda x : \tau.\ \ldots)$ and $t_1\ t_2$	$(\lambda\, x \in [\![\tau]\!].\ \ldots)$ and $t_1 \text{‘} t_2$
Polymorphic types	$\bigwedge$-quantification over sets
Types of variables $v : \tau$	Assumptions $[\![v]\!] \in [\![\tau]\!] \implies \ldots$
Proofs	Instrumented…

Universe

What is $\mathcal{U}$, the collection of all types?

Must be closed under $\Rightarrow$, etc., and admit choice!

Universe

What is $\mathcal{U}$, the collection of all types?

Must be closed under $\Rightarrow$, etc., and admit choice!

Sufficient: $V_{\omega+\omega} \setminus \{\emptyset\}$

We chose: All nonempty sets

Universe

What is $\mathcal{U}$, the collection of all types?

Must be closed under $\Rightarrow$, etc., and admit choice!

Sufficient: $V_{\omega+\omega} \setminus \{\emptyset\}$

We chose: All nonempty sets

Consequence: Need global choice axiom

$$\bigwedge A : \iota.\ [\, A \neq \emptyset \,] \Longrightarrow [\, \mathit{choose}\ A \in A \,]$$

Example

The transitivity rule

$$(\forall\alpha) \quad \bigwedge r\, s\, t : \alpha.\ [\, r = s \,] \Longrightarrow [\, s = t \,] \Longrightarrow [\, r = t \,]$$

Example

The transitivity rule

$$(\forall\alpha) \quad \bigwedge r\, s\, t : \alpha.\ [\, r = s\,] \Longrightarrow [\, s = t\,] \Longrightarrow [\, r = t\,]$$

is translated to

$$\bigwedge A : \iota.\ [\, A \neq \emptyset\,] \Longrightarrow (\bigwedge r : \iota.\ [\, r \in A\,] \Longrightarrow (\bigwedge s : \iota.\ [\, s \in A\,] \Longrightarrow$$
$$(\bigwedge t : \iota.\ [\, t \in A\,] \Longrightarrow [\langle\, r =_A s\,\rangle] \Longrightarrow [\langle\, s =_A t\,\rangle] \Longrightarrow [\langle\, r =_A t\,\rangle])))$$

Example

The transitivity rule

$$(\forall\alpha) \quad \bigwedge r\, s\, t : \alpha.\ [\, r = s\,] \Longrightarrow [\, s = t\,] \Longrightarrow [\, r = t\,]$$

is translated to

$$\bigwedge A : \iota.\ [\, A \neq \emptyset\,] \Longrightarrow (\bigwedge r : \iota.\ [\, r \in A\,] \Longrightarrow (\bigwedge s : \iota.\ [\, s \in A\,] \Longrightarrow$$
$$(\bigwedge t : \iota.\ [\, t \in A\,] \Longrightarrow [\langle\, r =_A s\,\rangle] \Longrightarrow [\langle\, s =_A t\,\rangle] \Longrightarrow [\langle\, r =_A t\,\rangle])))$$

which is equivalent to

$$\bigwedge A\, r\, s\, t : \iota.\ [\, A \neq \emptyset\,] \Longrightarrow [\, r \in A\,] \Longrightarrow [\, s \in A\,] \Longrightarrow [\, t \in A\,]$$
$$\Longrightarrow [\langle\, r =_A s\,\rangle] \Longrightarrow [\langle\, s =_A t\,\rangle] \Longrightarrow [\langle\, r =_A t\,\rangle]\ .$$

Types and Terms

Translation of types:

$$
\begin{aligned}
[\![\, \kappa\ \overline{\tau_n} \,]\!] &:= \widehat{\kappa}\ \overline{[\![\, \tau_m \,]\!]} \\
[\![\, \alpha \,]\!] &:= x_\alpha
\end{aligned}
$$

Translation of terms:

$$
\begin{aligned}
[\![\, c[\overline{\tau_n}] \,]\!] &:= \widehat{c}\ \overline{[\![\, \tau_m \,]\!]} \\
[\![\, \lambda x : \tau.\ t \,]\!] &:= \lambda x \in [\![\, \tau \,]\!] .\ [\![\, t \,]\!] \\
[\![\, x \,]\!] &:= x \\
[\![\, t_1\ t_2 \,]\!] &:= [\![\, t_1 \,]\!] \text{ ` } [\![\, t_2 \,]\!]
\end{aligned}
$$

Translation of outer proposition structure:

$$
\begin{aligned}
[\![\, (\forall \overline{\alpha_m})\ \ \phi \,]\!] &:= \bigwedge \overline{x_{\alpha_m}} : \iota.\ \overline{[\, x_{\alpha_m} \neq \emptyset \,]} \Longrightarrow [\![\, \phi \,]\!] \\
[\![\, \bigwedge x : \tau.\ \phi \,]\!] &:= \bigwedge x : \iota.\ [\, x \in [\![\, \tau \,]\!] \,] \Longrightarrow [\![\, \phi \,]\!] \\
[\![\, \phi_1 \Longrightarrow \phi_2 \,]\!] &:= [\![\, \phi_1 \,]\!] \Longrightarrow [\![\, \phi_2 \,]\!] \\
[\![\, [\, t \,] \,]\!] &:= [\langle\, [\![\, t \,]\!] \,\rangle]
\end{aligned}
$$

Proofs

$$
\begin{array}{rcl}
[\![\lambda x : \tau .\ p]\!] & := & \lambda x : \iota .\ \lambda h : [x \in [\![\tau]\!]].\ [\![p]\!] \\
[\![\lambda h : \phi .\ p]\!] & := & \lambda h : [\![\phi]\!] .\ [\![p]\!] \\
[\![h]\!] & := & h \\
[\![p_1 \bullet p_2]\!] & := & [\![p_1]\!] \bullet_{\mathrm{n}} [\![p_2]\!] \\
[\![p \odot t]\!] & := & [\![p]\!] \odot [\![t]\!] \bullet_{\mathrm{n}} \{ [\![t]\!] \in [\![\tau]\!] \} \quad \text{where } t : \tau \\
[\![\lambda \overline{\alpha_m} .\ p]\!] & := & \lambda \overline{x_{\alpha_m}} : \iota .\ \lambda \overline{h_m : [x_{\alpha_m} \neq \emptyset]} .\ [\![p]\!] \\
[\![thm[\overline{\tau_n}]]\!] & := & \widehat{thm} \odot \overline{[\![\tau_m]\!]} \bullet_{\mathrm{n}} \overline{\{ [\![\tau_m]\!] \neq \emptyset \}}
\end{array}
$$

Proofs

$$
\begin{aligned}
[\![\, \lambda\, x : \tau.\, p \,]\!] &:= \lambda\, x : \iota.\, \lambda\, h : [\, x \in [\![\, \tau \,]\!] \,].\ [\![\, p \,]\!] \\
[\![\, \lambda\, h : \phi.\, p \,]\!] &:= \lambda\, h : [\![\, \phi \,]\!] .\ [\![\, p \,]\!] \\
[\![\, h \,]\!] &:= h \\
[\![\, p_1 \bullet p_2 \,]\!] &:= [\![\, p_1 \,]\!] \bullet_{\mathrm{n}} [\![\, p_2 \,]\!] \\
[\![\, p \odot t \,]\!] &:= [\![\, p \,]\!] \odot [\![\, t \,]\!] \bullet_{\mathrm{n}} \{[\![\, t \,]\!] \in [\![\, \tau \,]\!]\} \quad \text{where } t : \tau \\
[\![\, \lambda\, \overline{\alpha_m}.\, p \,]\!] &:= \lambda\, \overline{x_{\alpha_m}} : \iota.\, \lambda\, \overline{h_m : [\, x_{\alpha_m} \neq \emptyset \,]}.\ [\![\, p \,]\!] \\
[\![\, thm[\overline{\tau_n}] \,]\!] &:= \widehat{thm} \odot \overline{[\![\, \tau_m \,]\!]} \bullet_{\mathrm{n}} \overline{\{[\![\, \tau_m \,]\!] \neq \emptyset\}}
\end{aligned}
$$

$\{P\}$: Placeholder for proof of P (by tactics)

Proofs

$$
\begin{array}{rcl}
[\![\lambda x : \tau.\ p]\!] & := & \lambda x : \iota.\ \lambda h : [x \in [\![\tau]\!]].\ [\![p]\!] \\
[\![\lambda h : \phi.\ p]\!] & := & \lambda h : [\![\phi]\!].\ [\![p]\!] \\
[\![h]\!] & := & h \\
[\![p_1 \bullet p_2]\!] & := & [\![p_1]\!] \bullet_n [\![p_2]\!] \\
[\![p \odot t]\!] & := & [\![p]\!] \odot [\![t]\!] \bullet_n \{[\![t]\!] \in [\![\tau]\!]\} \quad \text{where } t : \tau \\
[\![\lambda \overline{\alpha_m}.\ p]\!] & := & \lambda \overline{x_{\alpha_m}} : \iota.\ \lambda \overline{h_m : [x_{\alpha_m} \neq \emptyset]}.\ [\![p]\!] \\
[\![thm[\overline{\tau_n}]]\!] & := & \widehat{thm} \odot \overline{[\![\tau_m]\!]} \bullet_n \overline{\{[\![\tau_m]\!] \neq \emptyset\}}
\end{array}
$$

$\{P\}$	:	Placeholder for proof of P	(by tactics)
$\bullet_n$	:	On-the-fly $\beta\eta$-normalization	(by simplifier)

Overloading (by example)

$$\mathit{plus}[\mathit{nat}] :\equiv \mathit{nat\text{-}plus}$$

$$\mathit{plus}[\alpha \times \beta] :\equiv \lambda\, x\, y : \alpha \times \beta.\ (\mathit{plus}[\alpha]\ (\mathit{fst}\ x)\ (\mathit{fst}\ y), \mathit{plus}[\beta]\ (\mathit{snd}\ x)\ (\mathit{snd}\ y))$$

Overloading (by example)

$$\mathit{plus}[\mathit{nat}] :\equiv \mathit{nat\text{-}plus}$$
$$\mathit{plus}[\alpha \times \beta] :\equiv \lambda\, x\, y : \alpha \times \beta.\ (\mathit{plus}[\alpha]\ (\mathit{fst}\ x)\ (\mathit{fst}\ y), \mathit{plus}[\beta]\ (\mathit{snd}\ x)\ (\mathit{snd}\ y))$$

Dictionary parameters:

$$\mathit{plus}_1 :\equiv \mathit{nat\text{-}plus}$$
$$\mathit{plus}_2[\alpha, \beta] :\equiv (\lambda\, (D_1 : \alpha \Rightarrow \alpha \Rightarrow \alpha)\, (D_2 : \beta \Rightarrow \beta \Rightarrow \beta)\, (x\, y : \alpha \times \beta).$$
$$(D_1\ (\mathit{fst}\ x)\ (\mathit{fst}\ y), D_2\ (\mathit{snd}\ x)\ (\mathit{snd}\ y)))$$

Overloading (by example)

$$plus[nat] :\equiv nat\text{-}plus$$

$$plus[\alpha \times \beta] :\equiv \lambda\, x\, y : \alpha \times \beta.\ (plus[\alpha]\ (fst\ x)\ (fst\ y), plus[\beta]\ (snd\ x)\ (snd\ y))$$

Dictionary parameters:

$$plus_1 :\equiv nat\text{-}plus$$

$$plus_2[\alpha, \beta] :\equiv (\lambda\, (D_1 : \alpha \Rightarrow \alpha \Rightarrow \alpha)\, (D_2 : \beta \Rightarrow \beta \Rightarrow \beta)\, (x\, y : \alpha \times \beta).\ (D_1\ (fst\ x)\ (fst\ y), D_2\ (snd\ x)\ (snd\ y)))$$

Translation of a simple theorem:

$$[\![\, comm[\alpha]\ plus[\alpha] \Longrightarrow comm[\beta]\ plus[\beta] \Longrightarrow comm[\alpha \times \beta]\ plus[\alpha \times \beta]\,]\!]$$

Overloading (by example)

$$plus[nat] :\equiv nat\text{-}plus$$
$$plus[\alpha \times \beta] :\equiv \lambda\, x\, y : \alpha \times \beta.\ (plus[\alpha]\ (fst\ x)\ (fst\ y), plus[\beta]\ (snd\ x)\ (snd\ y))$$

Dictionary parameters:

$$plus_1 :\equiv nat\text{-}plus$$
$$plus_2[\alpha, \beta] :\equiv (\lambda\, (D_1 : \alpha \Rightarrow \alpha \Rightarrow \alpha)\, (D_2 : \beta \Rightarrow \beta \Rightarrow \beta)\, (x\, y : \alpha \times \beta).$$
$$(D_1\ (fst\ x)\ (fst\ y), D_2\ (snd\ x)\ (snd\ y)))$$

Translation of a simple theorem:

$$[\![\ comm[\alpha]\ plus[\alpha] \Longrightarrow comm[\beta]\ plus[\beta] \Longrightarrow comm[\alpha \times \beta]\ plus[\alpha \times \beta]\ ]\!]$$
$$= \bigwedge D_1 D_2.\ comm[\alpha]\ D_1 \Longrightarrow comm[\beta]\ D_2 \Longrightarrow comm[\alpha \times \beta]\ (plus_2[\alpha, \beta]\ D_1\ D_2)$$

Overloading vs. Type Definitions

$$\textbf{typedef}\ \textit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$$

Overloading vs. Type Definitions

$$\textbf{typedef}\ \textit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \le y \longrightarrow f\ x \le f\ y)$$

Type *mfun* depends on overloaded operation $\le$!!!

Overloading vs. Type Definitions

$$\textbf{typedef}\ \mathit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \le y \longrightarrow f\ x \le f\ y)$$

Type *mfun* depends on overloaded operation $\le$!!!

Dictionary translation would introduce dependent types

Overloading vs. Type Definitions

$$\textbf{typedef}\ \textit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \leq y \longrightarrow f\ x \leq f\ y)$$

Type *mfun* depends on overloaded operation $\leq$!!!

Dictionary translation would introduce dependent types

Not noticed by [Wenzel ’97] [Haftmann and Wenzel ’06] [Obua ’06]

Overloading vs. Type Definitions

$$\textbf{typedef}\ \textit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \le y \longrightarrow f\ x \le f\ y)$$

Type *mfun* depends on overloaded operation $\le$!!!

Dictionary translation would introduce dependent types

Not noticed by [Wenzel '97] [Haftmann and Wenzel '06] [Obua '06]

Solutions:

- Disallow (but occurs in practice)

Overloading vs. Type Definitions

$$\textbf{typedef}\ \mathit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \le y \longrightarrow f\ x \le f\ y)$$

Type *mfun* depends on overloaded operation $\le$!!!

Dictionary translation would introduce dependent types

Not noticed by [Wenzel '97] [Haftmann and Wenzel '06] [Obua '06]

Solutions:

- Disallow (but occurs in practice)
- Eliminate overloading only in set theory

Overloading vs. Type Definitions

$$\textbf{typedef}\ \mathit{mfun}\ \alpha\ \beta \cong (\lambda f : \alpha \Rightarrow \beta.\ \forall x\ y.\ x \le y \longrightarrow f\ x \le f\ y)$$

Type *mfun* depends on overloaded operation $\le$!!!

Dictionary translation would introduce dependent types

Not noticed by [Wenzel '97] [Haftmann and Wenzel '06] [Obua '06]

Solutions:

- Disallow (but occurs in practice)
- Eliminate overloading only in set theory
- Unroll the type definition

Conclusion

Really spelled out the details of HOL + type classes + overloading

Conclusion

Really spelled out the details of HOL + type classes + overloading

“Implemented Semantics” uncovers foundational issues

Conclusion

Really spelled out the details of HOL + type classes + overloading

“Implemented Semantics” uncovers foundational issues

Facilitates experiments

Conclusion

Really spelled out the details of HOL + type classes + overloading

“Implemented Semantics” uncovers foundational issues

Facilitates experiments

Next step: Implicit arguments for ZF?